ARL-TR-8337 ● Mar 2018

ARL
US Army Research Laboratory

Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense

by Alexander Kott, Luigi V Mancini, Paul Théron, Martin Drašar, Edlira Dushku, Heiko Günther, Markus Kont, Benoît LeBlanc, Agostino Panico, Mauno Pihelgas, and Krzysztof Rzadca

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8337 ● MAR 2018

US Army Research Laboratory

Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense

by Alexander Kott
Office of the Director, ARL

Luigi V Mancini, Edlira Dushku, and Agostino Panico
Sapienza Università di Roma, Italy

Paul Théron
Thales, Paris, France

Martin Drašar
Masaryk University, Brno, Czech Republic

Heiko Günther
Fraunhofer, Wachtberg, Germany

Markus Kont and Mauno Pihelgas
NATO Cooperative Cyber Defence Centre of Excellence, Tallinn, Estonia

Benoît LeBlanc
Ecole Nationale Supérieure de Cognitique, Bordeaux, France

Krzysztof Rzadca
Institute of Informatics, University of Warsaw, Warsaw, Poland

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* March 2018	**2. REPORT TYPE** Technical Report	**3. DATES COVERED (From - To)** 9 September 2016–9 February 2018

4. TITLE AND SUBTITLE Initial Reference Architecture of an Intelligent Autonomous Agent for Cyber Defense	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Alexander Kott, Luigi V Mancini, Paul Théron, Martin Drašar, Edlira Dushku, Heiko Günther, Markus Kont, Benoît LeBlanc, Agostino Panico, Mauno Pihelgas, and Krzysztof Rzadca	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory Army Research Laboratory (ATTN: RDRL-D) 2800 Powder Mill Road, Adelphi, MD 20783-1138	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-8337
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) NATO Science and Technology Organisation Collaboration Support Office (CSO) BP 25, 92201 Neuilly sur Seine, France	**10. SPONSOR/MONITOR'S ACRONYM(S)** NATO
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NATO IST-152-RTG

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes an initial reference architecture for intelligent software agents performing active, largely autonomous cyber defense actions on military networks of computing and communicating devices. The report is produced by the North Atlantic Treaty Organization (NATO) Research Task Group (RTG) IST-152 "Intelligent Autonomous Agents for Cyber Defense and Resilience". In a conflict with a technically sophisticated adversary, NATO military tactical networks will operate in a heavily contested battlefield. Enemy software cyber agents—malware—will infiltrate friendly networks and attack friendly command, control, communications, computers, intelligence, surveillance, and reconnaissance and computerized weapon systems. To fight them, NATO needs artificial cyber hunters—intelligent, autonomous, mobile agents specialized in active cyber defense. With this in mind, in 2016, NATO initiated RTG IST-152. Its objective is to help accelerate development and transition to practice of such software agents by producing a reference architecture and technical roadmap. This report presents the concept and architecture of an Autonomous Intelligent Cyber Defense Agent (AICA). We describe the rationale of the AICA concept, explain the methodology and purpose that drive the definition of the AICA Reference Architecture, and review some of the main features and challenges of the AICA.

15. SUBJECT TERMS

intelligent agent, autonomy, cyber warfare, cyber security, agent architecture

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Alexander Kott
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UU	78	**19b. TELEPHONE NUMBER (Include area code)**
Unclassified	Unclassified	Unclassified			(301) 394-1507

Contents

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

Lead Author: Alexander Kott

This report describes an initial reference architecture for intelligent software agents performing active, largely autonomous cyber defense actions on military networks of computing and communicating devices. The report is produced by the North Atlantic Treaty Organization (NATO) Research Task Group (RTG) IST-152 "Intelligent Autonomous Agents for Cyber Defense and Resilience".

1.1 Objective

In a conflict with a technically sophisticated adversary, NATO military tactical networks will operate in a heavily contested battlefield. Enemy software cyber agents—malware—will infiltrate friendly networks and attack friendly command, control, communications, computers, intelligence, surveillance, and reconnaissance (C4ISR) and computerized weapon systems. To fight them, NATO needs artificial cyber hunters—intelligent, autonomous, mobile agents specialized in active cyber defense.

With this in mind, in 2016, NATO initiated RTG IST-152 "Intelligent Autonomous Agents for Cyber Defense and Resilience". Its objective is to help accelerate development and transition to practice of such software agents by producing a reference architecture and technical roadmap.

If such research is successful, it will lead to technologies that enable the following vision. NATO agents will stealthily patrol the networks, detect the enemy agents while remaining concealed, and then destroy or degrade the enemy malware. They will do so mostly autonomously, because human cyber experts will be always scarce on the battlefield. They will be adaptive because enemy malware is constantly evolving. They will be stealthy because the enemy malware will try to find and kill them. At this time, such capabilities remain unavailable for the defensive purposes of NATO. The IST-152 group is using a comprehensive, focused technical analysis to produce a first-ever reference architecture and technical roadmap for autonomous cyber defense agents. In addition, the RTG is working to identify and demonstrate selected elements of such capabilities, which are beginning to appear in academic and industrial research.

Scientists and engineers from several NATO nations have brought unique expertise to this project. Examples of such areas of expertise include software agents, automated strategic reasoning, self-management of security, deep learning, innovative architecture for active defense, and so on. Only by combining multiple areas of distinct expertise and a realistic and comprehensive approach can such a complex software agent be provided.

The output of the RTG will become a tangible starting point for acquisition activities by NATO nations. If based on a common reference architecture, software agents developed or purchased by different nations will be far more likely to be interoperable. Deployed on NATO networks, the software agents will become a massive force multiplier: the agents will augment the inevitably limited capabilities of human cyber defenders and will team with humans when ordered to do so. Without such agents, effective defense of NATO computer networks and systems will be impossible.

Without autonomous, intelligent cyber defense agents, NATO C4ISR will not survive an encounter with a determined, technically sophisticated enemy. To acquire and successfully deploy such agents, in an interoperable manner, NATO nations must create a common technical vision—a reference architecture and a roadmap. By combining unique expertise from several nations, the IST-152 group delivers a technical foundation for accelerated development and procurement of such capabilities.

1.2 Fundamental Choices and Assumptions

A key assumption taken by this report is that in a conflict with a technically sophisticated adversary, NATO military tactical networks will operate in a heavily contested battlefield. Enemy software cyber agents—malware—will infiltrate our networks and attack our C4ISR and computerized weapon systems, with a significant probability that cannot be ignored.

To focus the attention of our research group, we have chosen to limit the scope of our problem as follows. We consider a single military vehicle (either a combat or logistics vehicle) with one or more computers residing on the vehicle (see Section 3 for detailed discussion). Each computer contributes considerably to the operation of the vehicle or systems installed on the vehicle. One or more computers are assumed to have been compromised, where the compromise is either established as a fact or is suspected.

One of appropriate use cases could consider the operations of an autonomous cyber defense agent on a battle management system (BMS) computer on a combat vehicle.

Due to the contested nature of the communications environment (e.g., the enemy is jamming the communications or radio silence is required to avoid detection by the enemy), communications between the vehicle and other elements of the friendly force are limited and intermittent at best. Under some conditions, communications are entirely impossible.

Given the constraints on communications, conventional centralized cyber defense (i.e., an architecture where local sensors send cyber-relevant information to a central location where highly capable cyber defense systems and human analysts detect the presence of malware and initiate corrective actions remotely) is often infeasible. It is also unrealistic to expect that the human warfighters residing on the vehicle will have the necessary skills or time available to perform cyber defense functions locally on the vehicle, even more so if the vehicle is unmanned.

Therefore, cyber defense of the vehicle and its computing devices will be performed by an intelligent, autonomous software agent. The agent (or multiple agents per vehicle) will stealthily patrol the networks, detect the enemy agents while remaining concealed, and then destroy or degrade the enemy malware. The agent will have to do so mostly autonomously, without support or guidance by a human expert.

In most discussions in this report, the agent is considered as a monolithic piece of software. However, depending on the implementation, the agent's modules can be distributed over multiple processes or devices, or it could be implemented as a team of agents or subagents (some possibilities are discussed in Section 3).

To fight the enemy malware deployed on the friendly computer, the agent often has to take destructive actions, such as deleting or quarantining certain software. Such destructive actions are carefully controlled by the appropriate rules of engagement and are allowed only on the computer where the agent resides.

The actions of the agent, in general, cannot be guaranteed to preserve the availability or integrity of the functions and data of friendly computers. There is a risk that an action of the agent will "break" the friendly computer, disable important friendly software, or corrupt or delete important data. Developers of the agent will attempt to design its actions and planning capability to minimize the risk. This risk, in a military environment, has to be balanced against the death or destruction caused by the enemy if the agent's action is not taken.

Provisions are made to enable a remote or local human controller to fully observe, direct, and modify the actions of the agent. However, it is recognized that human control is often impossible. The agent, therefore, is able to plan, analyze, and perform most or all of its actions autonomously.

Similarly, provisions are made for the agent to collaborate with other agents (who reside on other computers); however, in most cases, because the communications are impaired or observed by the enemy, the agent operates alone.

The enemy malware, specifically, its capabilities and tactics, techniques, and procedures (TTPs), evolves rapidly. Therefore, the agent is capable of autonomous learning. Because the enemy malware knows that the agent exists and is likely to be present on the computer, the enemy malware seeks to find and destroy the agent. Therefore, the agent possesses techniques and mechanisms for maintaining a degree of stealth, camouflage, and concealment. More generally, the agent takes measures that reduce the probability that the enemy malware will detect the agent. The agent is mindful of the need to exercise self-preservation and self-defense.

It is assumed here that the agent resides on a computer where it was originally installed by a human controller or authorized process. We do envision a possibility that an agent may move itself (or move a replica of itself) to another computer. However, such propagation is assumed to occur only under exceptional and well-specified conditions, and takes place only within a friendly network—from one friendly computer to another friendly computer. (This is a controversial topic and we do not wish to derail our efforts by debating it.)

Here is a good place to mention the controversy about "good viruses". Such viruses have been proposed and angrily dismissed earlier (Muttik 2016). These criticisms do not apply here. This agent is not a virus, because it does not propagate except under explicit conditions within authorized and cooperative nodes. It is also used only in military environments, where most usual concerns do not apply.

1.3 Basic Concepts and Terminology

In this report, the term "agent" denotes software or a collection of software that resides and operates on one or more computing devices, perceives its environment, and executes purposeful actions on the environment (and on itself) to achieve the agent's goals. We use the following acronyms: the agent is the Autonomous Intelligent Cyber Defense Agent (AICA) and the architecture is the AICA Reference Architecture (AICARA).

The term "environment" here denotes everything that surrounds the agent and that the agent can perceive: the computer hardware and software where the agent operates, the vehicle, the enemy malware, the humans who communicate with the agent or with surrounding hardware and software, and other agents that this agent can find and with whom it can communicate.

The term "percept" denotes an element of information that the agent is able to obtain or receive; the percept reflects an attribute of the environment or a change in an attribute of the environment. The following are examples of percepts, partly inspired by De Gaspari et al. (2016):

- Report from Nmap probing
- Observation of a change to the file system
- A signal that someone has interacted with a fake webpage (honey-page) or fake service

The term “action” denotes any action that a software agent can execute on its environment. It can include an impact on other software or data, or a communication to a human or another agent. The following are examples of actions, partly inspired by De Gaspari et al. (2016):

- Remap ports.
- Check the integrity of the file system.
- Create and deploy a fake password file, with an alarm mechanism activated when the file is accessed.
- Create and deploy a fake webpage or web service.
- Deposit a file with a “poison pill”.
- Identify a suspicious file.
- Sandbox a suspicious file.
- Analyze the behavior of software in the sandbox.

Examples of actions and situations in which the agent takes such actions are described in Section 3.

The term “state” refers to a collection of values of the environment’s attributes. Generally, the state is not known either fully or accurately, and the agent must infer it, at least in part.

The term “plan” here refers to a sequence or a directed graph of actions that the agent generates in order to transform the current state of the environment into a different state more desirable by the agent. The plan can be conditional (i.e., it includes intermediate decisions based on the perceived state) or temporal (i.e., it includes constraints on when the actions are performed).

2. Architecture Overview

Lead Author: Paul Théron

Cyber defense agents considered in the AICARA can essentially do the following:

- Are capable of handling autonomously cyber threats affecting the perimeter they defend.
- Cooperate with one another when and as required and feasible.

Each agent is implemented within or in attachment to one delimited system or device. Cooperation between agents is achieved through available communications channels.

The AICARA, derived from Russell and Norvig (2009), is assumed to include the functional components outlined in Fig. 1.

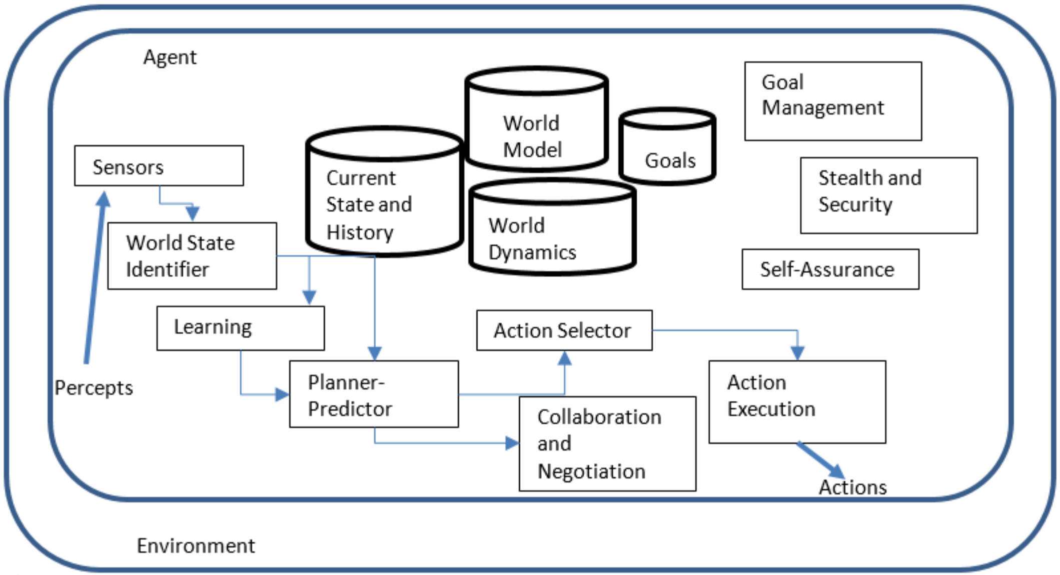


Fig. 1 Assumed functional architecture of the AICA

Note that at the time of publication, the AICA's functional architecture stands as an initial assumption and is discussed in later sections.

The AICA, through the AICARA, contributes the cyber defense of a military system or device through 5 main high-level functions (Fig. 2):

- Sensing and world state identification
- Planning and action selection
- Collaboration and negotiation

- Action execution
- Learning and knowledge improvement

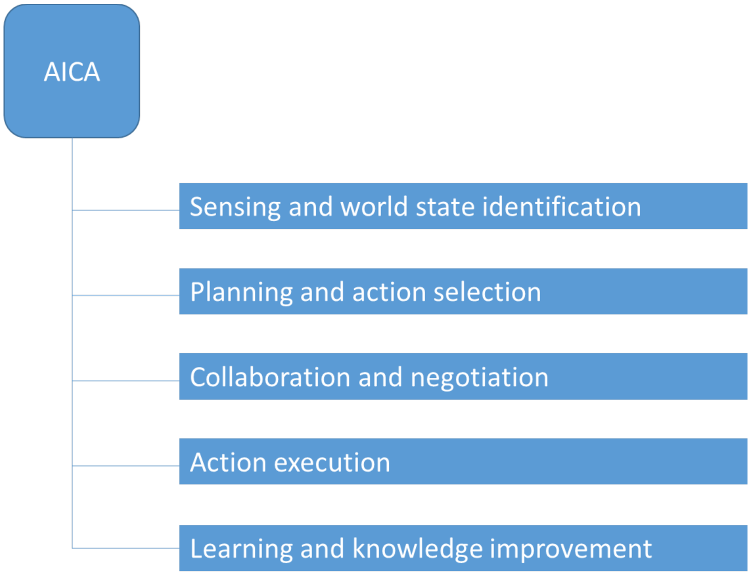

AICA
Sensing and world state identification
Planning and action selection
Collaboration and negotiation
Action execution
Learning and knowledge improvement

Fig. 2 The AICA's main 5 high-level functions

2.1 Sensing and World State Identification

Sensing and world state identification is the AICA high-level function that allows a cyber defense agent to acquire data from the environment and systems in which it operates, as well as from itself, to reach an understanding of the current state of the world and, should it detect risks in it, trigger the planning and action selection high-level function.

This high-level function relies upon the world model, current state and history, sensors, and world state identifier components of the assumed functional architecture.

It includes 2 functions:

- sensing
- world state identification

2.1.1 Sensing

Sensing operates from 2 types of data sources:

- external (system/device-related) current world state descriptors
- internal (agent-related) current state descriptors

Current world state descriptors, both external and internal, are captured on the fly by the agent's sensing function. They may be double checked, formatted, or normalized for later use by the world state identification function (to create processed current state descriptors).

2.1.2 World State Identification

The world state identification function operates from 2 sources of data:

- processed current state descriptors
- learned world state patterns

Learned world state patterns are stored in the agent's world knowledge repository. Processed current state descriptors and learned world state patterns are compared to identify problematic current world state patterns (i.e., presenting an anomaly or a risk). When identifying a problematic current world state pattern, the world state identification function triggers the planning and action selection high-level function.

2.2 Planning and Action Selection

Planning and action selection is the AICA high-level function that allows a cyber defense agent to elaborate one to several action proposals and propose them to the action selector function, which decides the action or set of actions to execute to resolve the problematic world state pattern previously identified by the world state identifier function.

This high-level function relies upon the world dynamics, actions and effects, goals, the actions' effect predictor, and action selector components of the assumed functional architecture.

It includes 2 functions:

- planning
- action selector

2.2.1 Planning

The planning function operates on the basis of 2 data sources:

- problematic current world state pattern
- repertoire of actions (response repertoire)

The problematic current world state pattern and repertoire of actions (response repertoire) are concurrently explored to determine the action or set of actions (proposed response plan) that can resolve the submitted problematic current world state pattern. The action or set of actions so determined are presented to the action selector.

It may be possible that the planning function requires some form of cooperation with other agents or a central cyber command and control (C2) to come up with an optimal set of actions forming a global response strategy. Such cooperation could be to either request from other agents or the cyber C2 complementary action proposals or delegate to the cyber C2 the responsibility of coordinating a global set of actions forming the wider response strategy. This aspect is not yet studied in the present release of the AICARA.

2.2.2 Action Selector

The action selector function operates on the basis of 3 data sources:

- proposed response plans
- agent's goals
- execution constraints and requirements (e.g., environment's technical configuration, and so on)

The proposed response plan is analyzed by the action selector function in the light of the agent's current goals, and the execution constraints and requirements that may either be part of the world state descriptors gained through the sensing and world state identifier high-level function or be stored in the agent's data repository and originated in the learning and knowledge improvement high-level function. The proposed response plan is then trimmed from whatever element does not fit the situation at hand and augmented by prerequisite, preparatory, precautionary, or postexecution recommended complementary actions. The action selector thus produces an executable response plan and then submitted to the action execution high-level function.

Like with the planning function, it is possible that the action selector function is required to liaise with other agents or a central cyber C2 to come up with an optimal executable response plan forming part of and being in line with a global response strategy. Such cooperation could be to exchange and consolidate information with other agents or the central cyber C2, and then agree collectively on the assignment of responsibilities over the various parts of the execution of the global executable response plan to specific agents. Alternatively, it could be to delegate to the cyber C2 the responsibility of elaborating a consolidated executable response plan and then assign to specific agents the responsibility of executing part(s) of the overall plan within their dedicated perimeter. This aspect is not yet studied in the present release of the AICARA.

2.3 Collaboration and Negotiation

Collaboration and negotiation is the AICA high-level function that allows a cyber defense agent to 1) exchange information (elaborated data) with other agents or a central cyber C2, for instance, when one of the agent's functions is not capable on its own of reaching satisfactory conclusions or usable results; and 2) negotiate with its partners the elaboration of a consolidated conclusion or result.

This high-level function relies upon coordination with other agents and the C2 component of the assumed functional architecture.

It includes, at the present stage, one function:

- collaboration and negotiation

The collaboration and negotiation function operates on the basis of 3 data sources:

- internal, outgoing data sets (i.e., sent to other agents or a central C2)
- external, incoming data sets (i.e., received from other agents or a central cyber C2)
- the agents' own knowledge (i.e., consolidated through the learning and knowledge improvement high-level function)

When an agent's planning and action selector function or other functions need it, the agent's collaboration and negotiation function is activated. Ad hoc data are sent to (selected?) agents or a central C2. The receiver(s) may or may not be able to elaborate further on the basis of the data received through their own collaboration and negotiation function. At this stage, when each agent (including possibly a central cyber C2) has elaborated further conclusions, it should share them with other (selected?) agents, including (or possibly not) the one that placed the original request for collaboration. Once this response (or these multiple responses) is received, the network of involved agents would start negotiating a consistent, satisfactory set of conclusions. Once an agreement reached, the concerned agent(s) could spark the next function within their own decision-making process.

When the agent's own security is threatened the agent's collaboration and negotiation function should help warning other agents (or a central cyber C2) of this state.

This release of the AICARA does not describe the agent's security monitoring and management.

Furthermore, the agent's collaboration and negotiation function may be used to receive warnings from other agents that may trigger the agent's higher state of alarm.

Finally, the agent's collaboration and negotiation function should help agents discover other agents and establish links with them.

This release of the AICARA does not describe nor specify the exchange protocol and the negotiation process, nor the alarm state raising mechanism and agent network discovery mechanism. These are issues to be further studied in later research and technology group meetings.

2.4 Action Execution

The action execution is the AICA high-level function that allows a cyber defense agent to effect the action selector function's decision about an executable response plan (or the part of a global executable response plan assigned to the agent), monitor its execution and its effects, and provide agents with the means to adjust the execution of the plan (or possibly to dynamically adjust the plan) as and when needed.

This high-level function relies upon the goals and actuators components of the assumed functional architecture.

It includes 4 functions:

- action effector
- execution monitoring
- effects monitoring
- execution adjustment

2.4.1 Action Effector

The action effector function operates on the basis of 2 data sources:

- executable response plan
- environment's technical configuration

Taking into account the environment's technical configuration, the action effector function executes each planned action in the scheduled order.

2.4.2 Execution Monitoring

The execution monitoring operates on the basis of 2 data sources:

- executable response plan
- plan execution feedback

The execution monitoring function should be able to monitor (possibly through the sensing function) each action's execution status (for instance, done, not done, or wrongly done). Any status apart from "done" should trigger the execution adjustment function.

2.4.3 Effects Monitoring

The effects monitoring function operates on the basis of 2 data sources:

- executable response plan
- environment's change feedback

It should be able to capture (possibly through the sensing function) any modification occurring in the plan execution's environment. The associated data set should be analyzed/explored. The result of such data exploration might (should) provide a positive (satisfactory) or negative (unsatisfactory) environment change status. Should this status be negative, this should trigger the execution adjustment function.

2.4.4 Execution Adjustment

The execution adjustment function operates on the basis of 3 data sources:

- executable response plan
- plan execution feedback and status
- environment's change feedback and status

The execution adjustment function should explore the correspondence between the 3 data sets to find alarming associations between the implementation of the executable response plan and its effects. Should warning signs be identified, the execution adjustment function should either adapt the actions' implementation to circumstances or modify the plan.

The update of the response plan in the course of its execution is not studied in the current release of the AICARA. It presents actual issues that require extended research work.

2.5 Learning and Knowledge Improvement

Learning and knowledge improvement is the AICA high-level function that allows a cyber defense agent to use the agent's experience to improve progressively its efficiency with regard to all other functions.

This high-level function relies upon the learning and goals modification *components* of the assumed functional architecture.

It includes 2 functions:

- learning
- knowledge improvement

2.5.1 Learning

The learning function operates on the basis of 2 data sources:

- feedback data from the agent's functioning
- feedback data from the agent's actions

The learning function collects both data sets and analyzes the reward function of the agent (distance between goals and achievements) and their impact on the agent's knowledge database. Results feed the knowledge improvement function.

2.5.2 Knowledge Improvement

The knowledge improvement function operates on the basis of 2 data sources:

- results (propositions) from the learning function
- current elements of the agent's knowledge

The knowledge improvement function merges results (propositions) from the learning function and the current elements of the agent's knowledge.

The current release of the AICARA provides a basic description of the learning and knowledge improvement high-level function and discusses the role of artificial intelligence methods in this context.

2.6 Generic Agent's Process Flow

The overall functioning of an agent is summarized in the following graph representing the generic agent's process flow (Fig. 3).

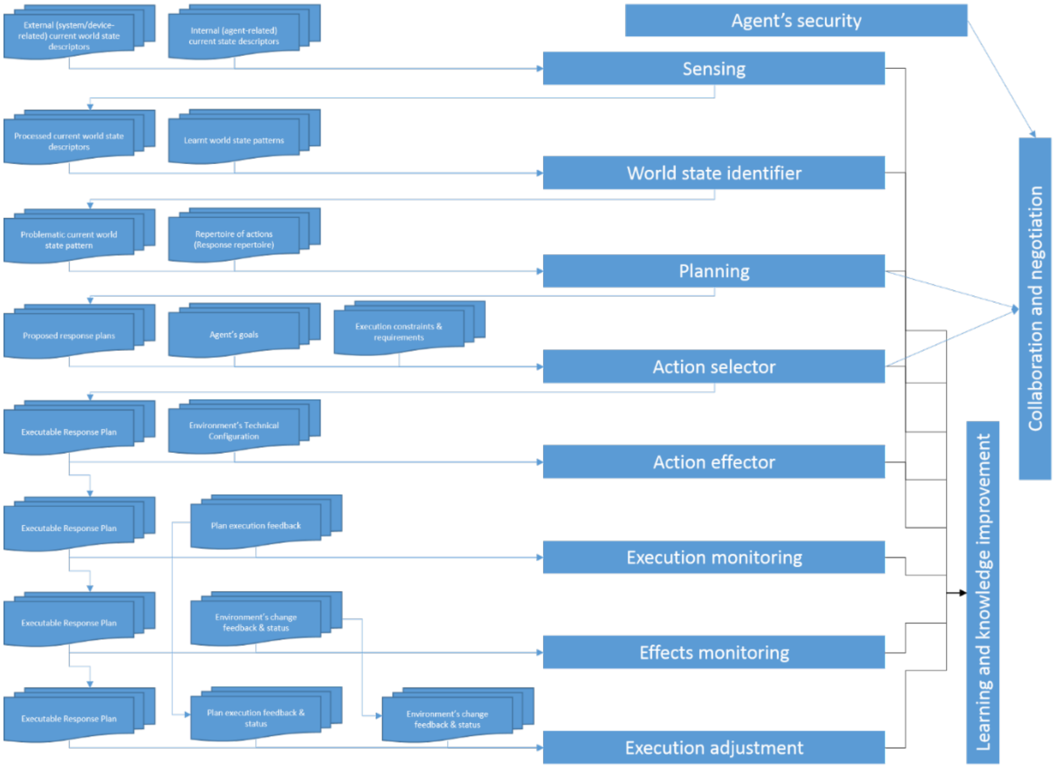

External (system/device-related) current world state descriptors
Internal (agent-related) current state descriptors
Agent's security
Sensing
Processed current world state descriptors
Learnt world state patterns
World state identifier
Problematic current world state pattern
Repertoire of actions (Response repertoire)
Planning
Proposed response plans
Agent's goals
Execution constraints & requirements
Action selector
Executable Response Plan
Environment's Technical Configuration
Action effector
Executable Response Plan
Plan execution feedback
Execution monitoring
Executable Response Plan
Environment's change feedback & status
Effects monitoring
Executable Response Plan
Plan execution feedback & status
Environment's change feedback & status
Execution adjustment
Collaboration and negotiation
Learning and knowledge improvement

Fig. 3 The generic AICA process flow

3. Scenarios

Lead Author: Heiko Günther

3.1 Context

We define a number of example scenarios to explain the possibilities of an agent approach using realistic threats in the military domain.

The architecture for our scenarios is a generic military vehicle, carrying different IT components all connected to a vehicle network. Even though not all vehicles have this large number of components and the components may not be connected over just one network, it is a reasonable structure for describing attack scenarios against current and future IT equipment in military vehicles.

The following IT components are used in the vehicle (Fig. 4):

- **Switch (SW)**: This box describes a component that is able to interconnect different devices, not relying on a specific technology.

Approved for public release; distribution is unlimited.

- **Internal communication system (InterCOM):** This box is a device that enables users who are physically separated to communicate within the vehicle.

- **Weapon system (WS):** This is a symbolic box for the weapon systems on the vehicle. A real weapon system may be composed of different IT components, which we aggregate in this box.

- **Communication system (COMM)**: This box describes the communication systems between the vehicle and the external world (satellite communication, radio communication, etc.).

- **Electronic warfare system (EW):** This is a symbolic box for all the equipment that can be used to conduct electronic warfare (e.g., jamming).

- **Vehicle management system (VMS):** This box describes the internal management system of the vehicle. The VMS receives input from all the sensors around the vehicle, elaborates the data, and delivers a state of the vehicle to the user or other components.

- **Optoelectronic system (OPT):** This box represents the systems that can be used to visually check the environment (camera, infrared).

- **User laptop:** This box is a device that is used be the operator to interact with the onboard systems.

- **Mission-specific system (MS):** This is a symbolic box for systems that can be used during a specific mission or from a specific kind of user, such as an anti-mine system.

- **BMS:** This box represents the system that is used by the operators to gather information about their position on the field, the enemy position on the field, and the friendly forces positions. It is usually a geo-information system that relies on information from the positioning system. It gets information from the HQ and also updates the information in the HQ (and other vehicles connected to the HQ) by sending its own position or the position of identified entities.

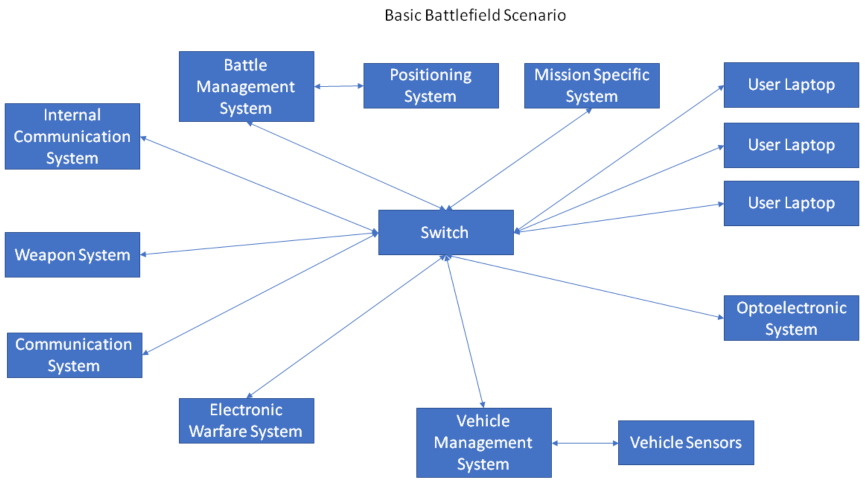


Fig. 4 Vehicle systems and network structure

3.2 Agent Deployment

Agents can be deployed in a centralized approach with master and slave agents or as a distributed network of self-organizing agents (Fig. 5).

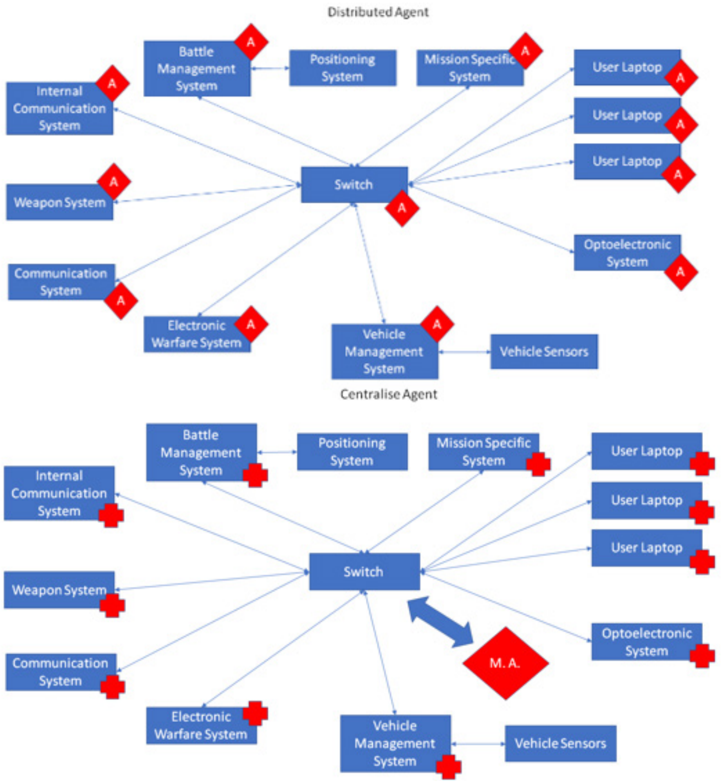


Fig. 5 Centralized agent network vs. distributed agent network

In a centralized approach, all the resource consuming evaluation of data can be done within the master agent. The master agent controls the slave agents and commands them to do actions. The slave agents, which have to be installed on military hardware, can be very simple (e.g., scripts that send data and execute commands). This approach is easier to implement in combination with military off-the-shelf equipment and may be a first step in the direction of agent-based security.

A distributed network of self-organizing agents is much more sophisticated. The agents have to share work, agree on a common situational picture, and decide together about further actions. It eliminates the master agent as a single point of failure and dramatically increases resilience. Even isolated or partitioned agents can continue to protect the system. This approach should be seen as the long-term goal for agent systems in the military domain.

3.2.1 Scenario A

The first scenario describes an advanced persistent threat (APT) targeting the integrity of the BMS (Table 1).

Table 1 APT targeting a BMS scenario

Attacker movement	Agent network activity
Stage 0: Primary Infection During the maintenance of the vehicle, malicious code is installed on the VMS. The VMS is now the footprint of the adversary in the system and is initiating an attack on the vehicle.	The agent network may not be able to detect the primary infection.
Stage 1: Reconnaissance The malware has to orient in the vehicle. The location of the BMS is not known to the VMS, so the malware starts probing for the ports commonly used by the BMS. In a second step, it scans the BMS for vulnerable software reachable from the VMS.	The agent network detects the scanning activity in the network. It puts the BMS on an alert state and decreases trust in the VMS.
Stage 2: Targeting and Delivery The malware maps the exploits it carries with the vulnerabilities on the BMS and delivers one of them.	The agent network detects the attack in the network traffic and tries to stop it. It sets BMS on high alert state and uses additional monitoring and active analysis to identify the status of the VMS and BMS. The agent network also limits the interaction possibilities of the VMS with the other components.
Stage 3: Exploitation and Installation The malware exploits a vulnerability on the BMS and places tempering software on the system.	The agent network detects the exploitation and learns that the countermeasures were not effective.
Stage 4: Actions The malware component on the BMS starts tempering information on the BMS. It modifies positions and friend/foe mappings. It tries to stay stealthy and uses the legitimate situation update function of the BMS to send the tempered information to the HQ. Doing this, it tempers the common situational picture of the HQ and the other vehicles.	The agent network restarts the BMS to reset it to a safe state. It isolates the VMS to prevent further infections of other components.

3.2.2 Scenario B

The second scenario adds an additional attack step to the first scenario (Table 2).

Table 2 APT targeting a BMS scenario with additional attack step

Attacker movement	Agent network activity
Stage 0: Scenario A The VMS is infected and the malware component on the BMS is tempering information on the BMS.	In that scenario, it is assumed that the agent network cannot reset the BMS and the attack continues.
Stage 1: Targeting The malware on the BMS knows the location of the COMM, because it is used for legitimate traffic with the HQ.	The agent network identifies anomalous traffic from the infected BMS to the COMM. It sets COMM on high alert state. It does not cut the connection because the BMS is mission critical.
Stage 2: Delivery, Exploitation, and Installation An exploit for the management system of the COMM is used to get control and modify the channel and encryption used by one of the radios. The device holding the connection to the HQ remains untouched to not lose the possibility to inject tempered information to the HQ BMS.	The agent network identifies that the COMM is behaving unusual and is under adversarial control. The agent network now enters an emergency state and creates a covered channel to send an alert to the HQ over the compromised COMM to inform it that it lost control and information from the vehicle is not integer.
Stage 3: C2 and Act The malware on the COMM informs the malware on the BMS about the successful reconfiguration of the radio. The malware on the BMS begins to exfiltrate the situational data over the tempered radio before it modifies the data.	The agent network does a mission impact assessment using predefined guidelines and reasoning to decide which components of the vehicle can be taken offline. It moves all agent network capabilities currently placed on compromised system to alternative locations. Finally, it executes the response plan and informs the commander which systems are not available anymore and where they have to switch to manual usage mode.

3.2.3 Scenario C

The third scenario describes an indirect infection of the WS (Table 3).

Table 3 Indirect infection of the WS scenario

Attacker movement	Agent network activity
Stage 1: Targeting and Delivery At the HQ, the laptop is infected by opening a malicious e-mail attachment or inserting an infected USB stick.	The agent network may not detect that attack, because the laptop is not connected to the vehicle network.
Stage 2: Exploitation and Installation When the malware detects that it is attached to the vehicle network, it escalates privileges and starts a second attack step.	The agent network identifies the privilege escalation and tries to quarantine the malware. It starts further active analysis and more sophisticated monitoring of the laptop.
Stage 3: Actions The malware tries to identify the WS by scanning the vehicle network for ports known to be used by the WS. It plans to use an exploit to take over the WS and temper the targeting information by some degree to decrease the effectiveness of the WS.	The agent network identifies anomalous traffic from the laptop that is under high alerted monitoring to the WS. It isolates the WS system from the laptop and stops the attack.

4. Data Services within Agents

Lead author: Paul Theron

This section describes the initial assumptions taken about the AICA's data services:

- world model
- world current state and history
- world dynamics knowledge

These modules of the agent are not just mere data repositories but producers of processed data (i.e., "information"). They embark on intelligence of their own or rely on external sources to produce information, possibly cooperating with other agent services for higher-order intelligence or support. Their communication with other agent services implies the definition of internal protocols. Their data must be protected. The agent's data services are built in a way similar to that of the diagram in Fig. 6.

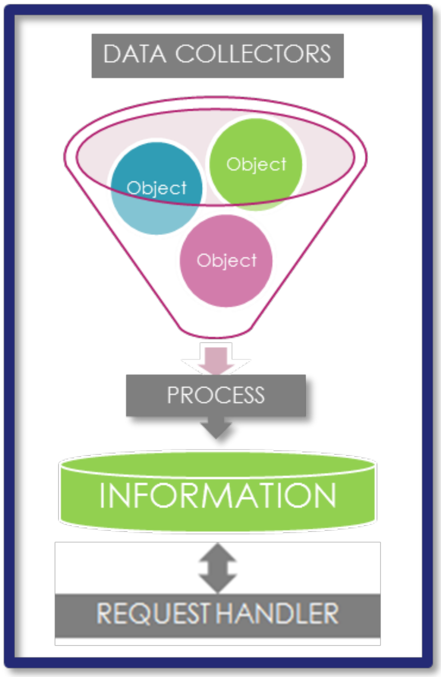


Fig. 6 General architecture of the data services

At the present stage, many options are open. We hypothesize the following:

- Data collectors accept incoming data records and check their compliance to formatting rules.
- Once verified, data records are processed. Processing may be limited to mere storage instructions or the data service module may have to perform more sophisticated data consolidation/aggregation functions.
- Data records can be requested by the agent's other modules. In this case, the data service's request handler might be designed to check the request against validity rules (according to agent design options), and then data are extracted, sorted, grouped, and bundled into an appropriate data container returned to the requesting module.

4.1 World Model Data Service

4.1.1 Definition

We hypothesize that a world model is the following:

- A formal descriptor of the elements it supplies to the agent's other services:
    - the nominal and degraded ontology of the agent

    - the nominal and degraded ontology of the system and environment (systems and threat)
    - the nominal and degraded patterns of the world's state (agent + environment + threat). Patterns express the agent, the system or its environment's static and dynamic relations, and the concurrency of their configurations.
- It is based on the following:
    - a theory of world models in the context of the cyber defense of military systems
    - a formal descriptive language
    - validated algorithms transforming inputs into descriptors
- Embedded into the agent, the model is one of the following:
    - calculated by the agent (which inflates the agent's size and requires computing power) or
    - loaded from external source (which requires periodic updates/uploads of data produced by external sources)

4.1.2 Inputs

We hypothesize that the world model data service may take the following classes of data as input:

- Data about the agent:
    - architecture, modules, and functions
    - processes and protocols
    - performance descriptors
- Data about the defended system:
    - identified vulnerabilities
    - security devices and barriers
    - topology of friendly agents network
    - connection components problems
    - hardware components problems

  - firmware components problems
  - operating system (OS) components problems
  - middleware components problems
  - applicative components problems
- Data about the defended system's environment:
  - sources of threats and attack C2 and tools
  - threat and vulnerability patterns (Common Attack Pattern Enumeration and Classification [CAPEC], Common Vulnerabilities and Exposures [CVEs], etc.)
  - indicators of compromise (IoCs) (OpenIOC, Malware Information Sharing Platform [MISP], etc.)
  - available cyber defense resources
  - surrounding systems
  - available resources for cyber defense.

The data sources would then be the following:

- cyber threat intelligence sources
- system descriptors (Simple Network Management Protocol [SNMP] data, packet-based switching [PBS], topology, configuration, etc.)
- the world state and history data service.

4.1.3 Process

There are 2 ways to produce ontologies and patterns of the world state:

- They can be created within the agent.
- They can be uploaded into the agent's database.

When created within the agent, input data are processed in the following ways:

- Collected through the agent's sensor (a standard format is required).
- Verified and preprocessed by the world model data service (e.g., normalized, formatted, and so on).
- Associated by the world model data service to form ontologies and patterns (ad hoc functionalities are required).

- Stored in the world model data service's database (a standard format is required).

4.1.4 Outputs

The hypothesized outputs of the agent's world data service are the following:

- Domain ontologies
    - agent
    - system
    - environment
    - threat
    - nominal and degraded
- World patterns
    - cross-domain patterns
    - domain-specific patterns
    - nominal and degraded

4.1.5 Current Issues and Lines of Research

Several issues can be identified at the present stage:

- the data classes required as input and the exact nature of output information
- the data formats of input data, data exchange protocols, and output information
- the algorithms for preprocessing, creating, and indexing data
- the type of the agent: should it be specialized (to process data) or standard?
- The risks to the agent's stealth due to the required memory size and processing power

4.2 World Current State and History

4.2.1 Definition

We hypothesize that the world current state is the evaluated distance between the world as it is and what it should be (based, for instance, on set goals). Pieces of information such as the following may be required to form state vectors/descriptors of the agent's world that can be used by the world state identifier module:

- nominal and degraded states of reference of agents, monitored systems, their environment, and threats
- memory of cyber defense actions and their impacts on the state of the world (in progress and past)
- current data about agents, monitored systems, environment, and threats

The world state identifier module can then be hypothesized to do the following:

- Calculate the current state data vector.
- Measure the deviation of the current state data vector from nominal state data vector.
- Interpret (meaning) of the deviation measure (based on history, actions in progress, etc.).
- Appraise (i.e., measure the valence of the interpretation).

The current world state is a formal descriptor of the appraised world's state at a given point in time and circumstances, usable by the world state identifier module.

The world state history is the chronological track record of world state descriptors.

4.2.2 Inputs

The world current state and history data service takes world state records from the world state identifier module.

4.2.3 Process

The world current state and history data service stores the new record provided by the state identifier module into its database.

4.2.4 Outputs

World state descriptor records are stored in the world current state and history's database.

4.2.5 Current Issues and Lines of Research

The issues identified at the present stage are the following:

- the specification of world state descriptors/vectors

- how the historical records of world state descriptors are used by the world state identifier
- the size of the world current state and history's database

4.3 World Dynamics Knowledge Data Service

4.3.1 Definition

We hypothesize that world dynamics are an object's behavioral rules and related expected states (nominal and degraded) in given circumstances.

They can be the following:

- a measure of how the world changes given its own parameters (defended systems, those systems' environments, and the threat on the systems and their environments)
- a measure of how the world changes given the agent's actions (a change in the status of the world)
- a measure of how agents change given their own parameters (a change of status or mode of the agent based on how it is designed)
- a measure of how the agent changes given the agent's and world's actions (a change in the status of the world)

Those changes can be measured through the following:

- states of reference (nominal and degraded) descriptors for agents, systems, environments, and threats
- agents and world entities' actions descriptors
- initial and final state descriptors

The world dynamics knowledge data service computes state transition patterns.

4.3.2 Inputs

We hypothesize that the world dynamics knowledge data service requires the following classes of data as input:

- data from the world model data service
- data about cyber threat dynamics:
    - behavior

    - expectations
    - change and factors
    - circumstances
- data about network and defense dynamics:
    - monitored and surrounding system(s)
        - behavior
        - expectations
        - change and factors
        - circumstances
    - agent itself and other friendly agents
        - behavior
        - expectations
        - change and factors
        - circumstances
    - incident response policy and mechanisms
        - behavior
        - expectations
        - change and factors
        - circumstances

4.3.3 Process

There are 2 possible ways to compute world state transition patterns out of input data:

- Data are processed live by the world dynamics knowledge data service.
- Data are uploaded into the world dynamics knowledge data service's database from externally provided records.

4.3.4 Outputs

The world dynamics knowledge data service computes state transition patterns.

4.3.5 Current Issues and Lines of Research

The complexity of the world (and even of the agent, as it is internally dynamic and adjusts to the world's changes) poses computational challenges. The second kind of technical challenges is related to the memory size and computation power required to produce state transition patterns.

5. Sensing and World State Identification

Lead Authors: Martin Drašar, Mauno Pihelgas, and Markus Kont

5.1 Overview

To interact with the world, the agent has to perceive and understand itself and its surroundings. This is accomplished by 2 subsystems: sensors and world state identifier. The sensors subsystem provides the agent with the means to observe itself and its environment, and communicate with other agents and C2 systems. The world state identifier then gives semantics to the received data by transforming them into a representation of a world in terms of a world model. It is also responsible for identifying environment changes, adversarial events, and anomalies in the sensory data.

The sensor system (Fig. 7) can contain a number of subsystems:

- **Self**: Used for monitoring the agent's memory and functions to ensure the agent's integrity.
- **System**: Used for monitoring system resources like memory, filesystem, and so on. It also monitors results of actions done by the agent. It can either be part of a monolithic agent or function as a separate module that feeds the agent data.
- **Environment**: Used for monitoring data coming from outside the agent. Can either be part of a monolithic agent or function as a separate module that feeds the agent data.
- **Comm**: Used for communication with other agents and C2.
- For the sensory data to be useful to the rest of the system, they should be properly normalized, correlated, fused, and deduplicated, so that only unique and relevant bits are passed on.

Approved for public release; distribution is unlimited.

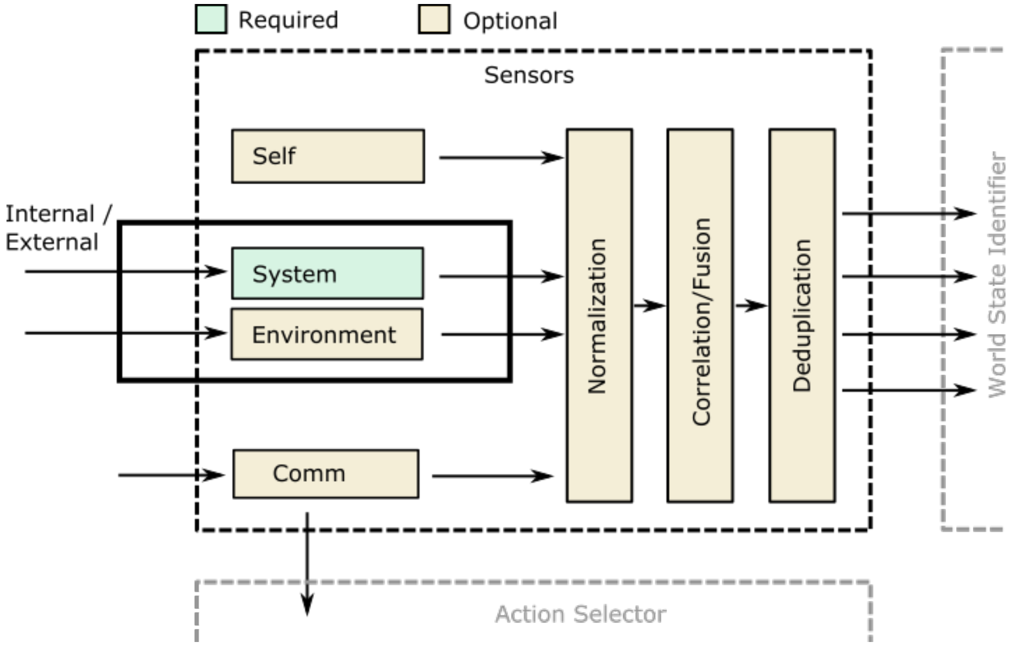


Fig. 7 Sensor system

The dataflow of the full-featured system includes the following:

- The sensors are responsible for gathering and processing data from both external and internal sources.
- First, to ensure continued operation, the sensors system monitors its internal health by collecting runtime statistics and checking their integrity.
- Furthermore, the input modules of the sensors fulfil the typical roles of system and network monitoring tools. The sensors collect logs and metrics from the others internal systems of the agent, the underlying host system (i.e., the OS), and relevant applications running on the host. The sensor is also capable of capturing network traffic from the host network interfaces. Alternatively, in case of a centralized agent, it is possible to capture traffic from a dedicated test access point device or monitoring port.
- The communications module enables communication with other agents and C2 via a secure channel. In case the agents are deployed on adversarial networks and the agents fail to establish secure communication channels, they may optionally fall back to using covert channels for communicating. The specification of the covert channel is beyond the scope of this reference architecture since the techniques depend highly on specific situations.
- To support a decentralized setup, the sensor system is also capable of receiving system and environment information from other agents.

- The data from the input modules always go through the input sanitation (normalization, correlation, fusion, and deduplication) process, which ensures that the data can be processed by other systems of the agent. This also applies to data received from other agents and C2, because the adversary may try to inject malicious or garbage data into the agent.

- The sensor system passes its data on to the world state identifier system.

The world state identifier (Fig. 8) processes sensor data to assess the state the world is in with respect to the world model. It consist of up to 4 processes:

- **Environment identification:** Based on the sensor data and the knowledge of expected world state, it identifies the environment the agent is running on. This process is mostly needed to distinguish running inside a virtual machine or inside a debugger to limit the adversary's ability to reverse engineer the agent.

- **Friend/foe identification:** Used mainly for identification and tagging of processes and files on the system. It is a prerequisite for offensive and defensive actions against adversaries as well as correct strategy planning.

- **Anomaly identification:** Used for detecting anomalies in data from sensors. The baseline for anomaly identification is encoded into the world state. The detection can be rule-based, pattern-based, or based on behavioral detection.

- **World state update:** Transforms sensor data and data from environment identification and friend/foe identification into a world model and world dynamics knowledge update.

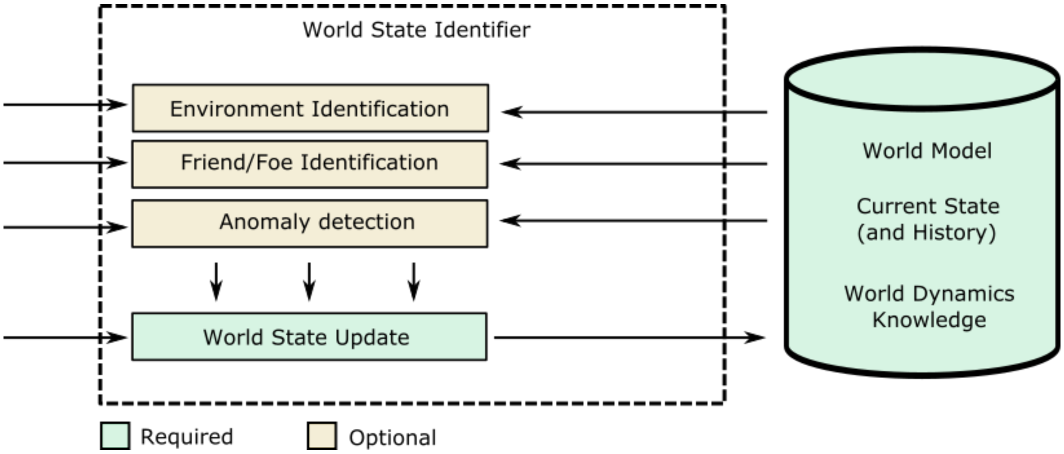


Fig. 8 World state identifier

The dataflow of the full-featured system includes the following:

- Updates on self and environment changes are received from the sensors.
- The world model, historical data and world dynamics knowledge are queried to have a baseline for sensor data processing.
- The environment identification component assesses any changes in the agent's environment.
- The friend/foe identification component identifies any potential adversaries and produces IoCs.
- The anomaly detection component estimates potentially anomalous behavior in sensory data and produces IoCs.
- Findings from the 3 previous components are combined with input sensory data and transformed to a world state update. This update is propagated to a world state database.

5.2 World Model and World Dynamics Knowledge

A world model is an abstraction of reality that provides a semantic meaning to perceived data. Its actual representation is strongly dependent on the implementation of the agent. In the optimal case, an agent is using data services to process, store, and employ sensory information transformed into the world model and world dynamics knowledge. These data services conform to the general description provided in Section 4 and are built with their own sets of constraints, which dictate their structure and capabilities.

In this section, we present a set of recommendations for a minimalistic world model structure and world dynamics knowledge, which are required for successful operation of an in-vehicle autonomous agent. Given the large amount of data the agent could be processing and the number of different states the agent could be in, the following should be satisfied:

- The model should use features based on the properties of the machines and network, which are normal during nonanomalous operation. Provided that most Army systems have precisely defined operation parameters, establishing a model as a baseline should be attainable.
- The model should encode explicit IoCs.
- The goals of the agent should be expressible as a function of a world state.

- Both the current state and world dynamics are also highly dependent on the agent's implementation and the design decisions for the model. Nevertheless, given the expected operational parameters of the agent, we suggest that the model used for the world and the current state should contain the components in Table 4. The world dynamics knowledge should be computed on the fly from the world model and the current state and history.

Table 4 Components of the world and current state and history models

Component	Model	Description
Flow database	Current state and history	Record of network flows, which can be augmented by full traffic traces where allowed by space constraints.
Log stash	Current state and history	Collection of system and application logs, preferably in a unified form suited for quick searching and analysis.
System metrics	Current state and history	Performance and operation characteristics of an agent and the system it is running on.
Whitelists	World model	Policies and baselines of normal behavior derived beforehand from the knowledge of the agent's environment.
Entity description	World model Current state and history	Both the description of entities in the agent's proximity and their current operational status as viewed by an agent (e.g., probability of compromise).

5.3 Use Case

To illustrate possible relations among the sensors, world state identifier, and world state, we present a slightly modified combination of Scenario A and Scenario B from Section 3. In this scenario, malicious code was inserted during maintenance to the VMS and manifests on the battlefield, propagating to the BMS and then to COMM.

The use case timeline and events are as follow:

1) The VMS gets infected during maintenance.

    a) Sensors (S): No information.

    b) World state identifier (I): No information.

    c) World state (W): No change.

2) Malware activates and attempts to infiltrate the BMS.

    a) S: Detected connection between the VMS and BMS.

b) I: Identified an anomalous connection and produced an IoC.

c) W: Updated with the IoC; the VMS and BMS are flagged as anomalously acting systems with potential to compromise.

3) BMS successfully compromised.

   a) S: The BMS supervising process identifies an integrity violation and logs the information.

   b) I: Logged information is transformed into an IoC.

   c) W: Updated with the IoC; the BMS is flagged as a potentially compromised system with higher confidence.

4) Malware attempts to infiltrate COMM.

   a) S: Detected connection between the VMS and COMM.

   b) I: Identified an anomalous connection and produced an IoC.

   c) W: Updated with the IoC; the VMS, and COMM are flagged as anomalously acting systems with potential to compromise, the VMS with higher confidence.

5) COMM successfully compromised.

   a) S: No information.

   b) I: No information.

   c) W: No change.

6) The VMS is functionally affected.

   a) S: Detected anomalies in vehicle responses.

   b) I: Anomaly report is converted into an IoC.

   c) W: Updated with the IoC; the VMS is flagged as compromised with the highest probability.

6. Action Selector and Predictor Modules

Authors: Benoît LeBlanc and Krzysztof Rzadca

6.1 Overview

We propose to decompose the decision-making process to decide the actions to do into 2 modules (Fig. 9): the planning (predictor) module and the action selector (selector) module.

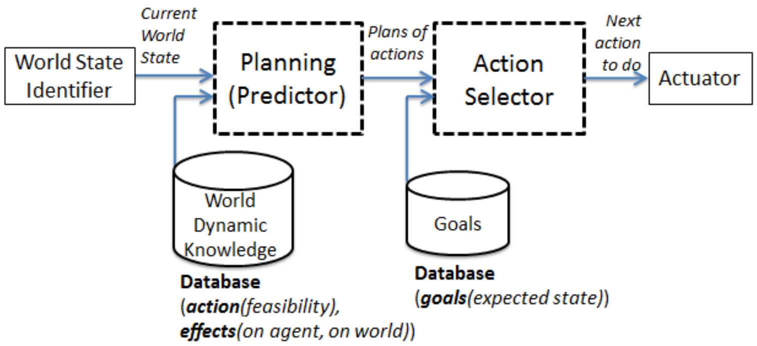


Fig. 9 Overview of the planning and selector modules

The goal of the predictor is to create a set of possible plans of actions that lead from the current world state to some possible future world states. By action we denote an activity that, once started, cannot be executed partially, in contrast with an action plan consisting of a sequence of actions. As a new plan is computed, the predictor sends it to the selector (in a semicontinuous process) and continues to compute alternative proposed plans.

The selector receives continuously proposed plans of actions leading to some future world states. Because of enemy actions or the world dynamics, there might be multiple future world states stemming from a single action in the first step. Aware of the goals, the selector choses an action plan that leads to the most desirable future world states and then sends the first actions from this plan to the action execution module (actuator). The selector orders the execution of just the first action; the subsequent actions are proposed in an updated plan by the predictor.

6.2 Planning Module (Predictor)

The predictor has access to a database of possible actions: a kind of a dictionary of all possible actions, including preconditions and prerequisites for each action. The predictor internally uses 2 functions to implement a tree exploration: a function f_a that maps the current world state (given by the world state identifier [WSI] module) to a set of feasible actions (i.e., a subset of the discussed database); and another function f_w that maps a world state and an action to a (set of) future world states (possibly with some information on the probability of individual states) (Fig. 10).

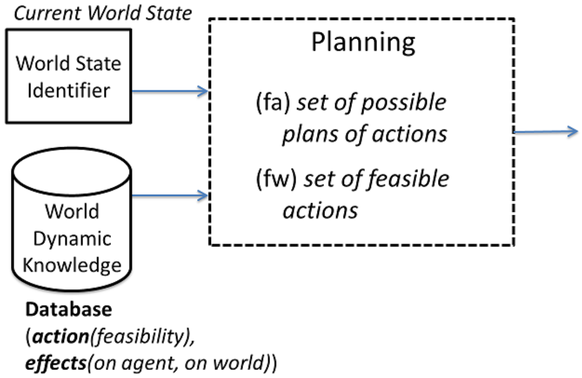


Fig. 10 Planning module (predictor)

Starting with the current state, the predictor uses f_a to produce a set of alternative actions. For instance, and greatly simplifying the situation, if the current state given by WSI module is (*vehicle engaged in combat*; and *a system file with changed SHA-1 hash*; and *previously unseen radio transmission detected*), the result of f_a, the set of feasible actions might be {no action, shut down radio comm Y, shut down the entire computer system}.

Then, on each of these actions, the predictor uses f_w, leading to a future world state (after the first action). For instance, the previous world system state combined with the action "shut down radio comm Y" may lead to the world state (*vehicle engaged in combat*; *a system file with changed SHA-1 hash*; and *weapon system Y malfunction*). The process of invoking f_a and f_w is continued, resulting in a tree; a leaf of this tree is a set of future world states (or a probabilistic distribution over this set).

Consider the following example fact and action databases.

Fact Database:

Fact 115, vehicle is engaged in an exercise
Fact 116, vehicle is on the parking
Fact 117, vehicle is engaged in a combat
Fact 223, system file with changed SHA-1 hash
Fact 225, system file with changed SHA-2 hash
Fact 312, weapon system Y functions correctly
Fact 314, weapon system Y malfunction is detected
Fact 451, usual radio transmission detected
Fact 452, rare but already noticed radio transmission detected
Fast 453, previously unseen radio transmission detected

A world state is composed by the conjunction of several facts, for instance,

[World State Identifier] → (State 322 = Fact 117 AND Fact 223 AND Fact 453)

This state is an input to the planning module. Consider the following possible actions.

Action Database:

Act 000, no action
Act 120, active radio comm Y
Act 121, shut down radio comm Y
Act 224, generate fake data file
Act 225, generate fake password file
Act 244, reinstall /etc from backup 12.345
Act 300, scan all ports
Act 310, remap ports
Act 384, open port 80
Act 390, close port 123
Act 420, kill process with PID 789
Act 499, shut down entire computer system

f_a(State 322) produces {Act 000, Act 121, Act 499} (Fig. 11).

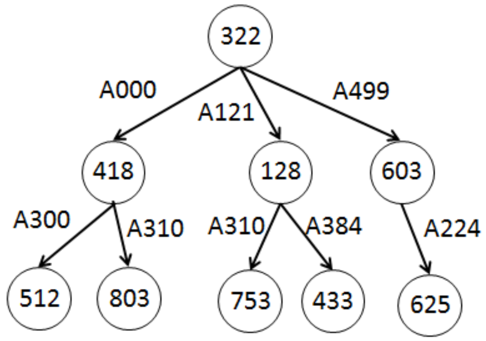


Fig. 11 A tree of possible actions

f_w(s322, a000) = State 418,
then f_a(State 418) produces {Act 300, Act 310}
f_w(s322, a121) = State 128,
then f_a(State 128) produces {Act 310, Act 384}
f_w(s322, a499) = State 603,
then f_a(State 603) produces {Act 224}
f_w(s418, a300) = State 512
f_w(s418, a310) = State 803
f_w(s128, a310) = State 753
f_w(s128, a384) = State 433
f_w(s603, a224) = State 625

When a path in the tree is calculated, it leads to a transmission to the selector module of a plan such as "(#plan, (action, state), (action, state), etc.)".

On the previous example, as soon as they are calculated, the predictor sends the following to the selector:

```
(001, (a000, s418), (a300, s512))
(002, (a000, s418), (a310, s803))
…
(005, (a499, s603), (a224, s625))
…
```

Functions f_a and f_w can be implemented by trained neural networks or rule-based systems. Both should be precalculated and implemented in the system. Updates could be done during the vehicle overhaul.

Function f_a is a service using the database "actions and effects": receiving the world state and returning a set of actions. Function f_w is a service of world dynamics knowledge, but we stress that f_w must consider not only the internal evolution of the world, but also the effects of a concrete action.

The predictor's algorithm effectively builds a complete search tree of the future actions and world states, and we acknowledge that such a tree probably must be pruned because of exponential search space.

An alternative to building a tree is to use a trained neural network directly. The input to the network is the current world state; the network produces future world states. Thus, the network implicitly implements f_a and $f_{w.}$

6.3 Selector

Based on the current world state, the selector decides whether the situation is urgent, for instance, when the vehicle is engaged in combat. Urgency corresponds to humans making reflex decisions in threat situations. In the urgent decision-making mode, one of the first action plans suggested by the predictor is chosen; otherwise, the selector may wait for a longer time for the predictor to send more actions plans.

The selector chooses an action plan based on how the predicted future world states match with the goals of the mission. A goal can be expressed through a function of the (future) world state, mapping the state into the degree this particular goal is fulfilled. We acknowledge that there might be multiple goals and that their relative importance might change depending on the current state of the mission. For instance, one goal might be to maintain the information integrity of the vehicle, another to keep the crew as safe as possible, and yet another to achieve the mission's principal, tactical objective.

The selector picks an action plan and then executes only the first action of the plan. The selector expects that, in the future, the predictor will send an updated action plan, taking into account the chosen action and the actual change observed in the world state. The selector must inform the predictor that it has chosen an action and sent it for execution. As the world state is then no longer valid, the predictor should stop generating new action plans.

An alternative design is that the selector chooses a plan, and then executes actions autonomously, without consulting the predictor (or alternatively it can interrupt a plan with another unrelated plan, proposed in the future by the predictor). However, this solution does not allow for refinement of a plan by the predictor: for instance, an action night lead to 3 different world states; when an actuator executes an action, the predictor can refine the plan based on the actual observed new world state.

On the previous example, the selector receives from the predictor the following sequence of plans:

```
(001, (a000, s418), (a300, s512))
(002, (a000, s418), (a310, s803))
…
(005, (a499, s603), (a224, s625))
```

After receiving plans 001 and 002, and due to the known agent's goals, the selector chooses plan 002 and send to the activator order to do Act000. The predictor continues to send alternative plans (plan 003, plan 004, etc.) until the WSI module sends a new world state. Then the predictor calculates from the new situation (which is state 418 in our example) that a300 and a310 are still possible and deep exploration of the future leads to actions to enchain after a300 or a310; plans 001 and 002 are refined, but plans 003 to 005 are dropped. If no unexpected thing has appeared, the predictor will send the following:

```
(001, (a300, s512), (a224, s788))
(006, (a300, s512), (a000, s789))
(002, (a310, s803), (a225, s901))
…
```

6.4 Examples

We use the standard use case of a military vehicle with several machines connected by an internal network.

6.4.1 A Standard Cyberattack

A standard cyberattack is a known attack that with a high probability would not lead to negative effects. For instance, it is an attack that tries to use a known bug that is patched in the version of the OS used by all the machines in the vehicle. Furthermore, we assume that there is no time urgency—for instance, the vehicle is parked at the base.

The world state catches the symptoms of the attack by the network sensors (e.g., flow of data or connection attempts to a certain port). The predictor constructs an action plan leading to a future world state in which the attack is attributed. The plan starts with placing on the machine small fake information (e.g., the first action is to create a mock-up password file and another is to create a file with a name suggesting classified content). In a future world state, after creating a mock-up password file, this password file is either accessed or not. If there is an access, the next action is to generate more mock-up password files. If the file is ignored, the next action is to generate a file that pretends to contain classified information. However, the predictor also proposes other action plans unrelated to the current attack (such as proposing actions executing order from the C2) or doing nothing.

As the vehicle is parked, the goal of attributing the attack is the most important. Thus, the selector choses action "small mock-up information" and orders the actuators to execute the first action (i.e., to generate a mock-up password file).

6.4.2 An Unexpected Cyberattack

An unexpected cyberattack is detected through its results, rather than by intercepting the attack as it happens. For instance, a routine file system check may detect a changed hash value of a system file. There might be also time urgency: the vehicle might be engaged in active combat. As such a situation is a threat to the integrity of the system, the predictor and selector must act quickly. The predictor suggests a short plan of, for example, restoring the changed file from a backup; the selector chooses this action based on the goal of maintaining integrity.

6.4.3 Cyber Exploration

If a vehicle is parked and the world state model does not detect an attack, the selector and predictor modules might use the opportunity to provide new data for the learning module. As the number of monitored characteristics of the world are vast, one of the important goals of the system is to be able to automatically distinguish a rare threat from a large number of normal, acceptable states. Similarly, given a large number of possible actions (closing a communication port, restoring a file, creating a file, etc.), the system must be able to learn the effects of the intended consequences of actions (e.g., closing TCP port 12345 would shut down the internal communication system XYZ).

During the cyber exploration scenario, the predictor would create action plans consisting of steps of basic actions (close TCP port 12345, open TCP port 12345) and observe their effects on the integrity of the vehicle.

7. Action Execution

Authors: Agostino Panico and Luigi Mancini

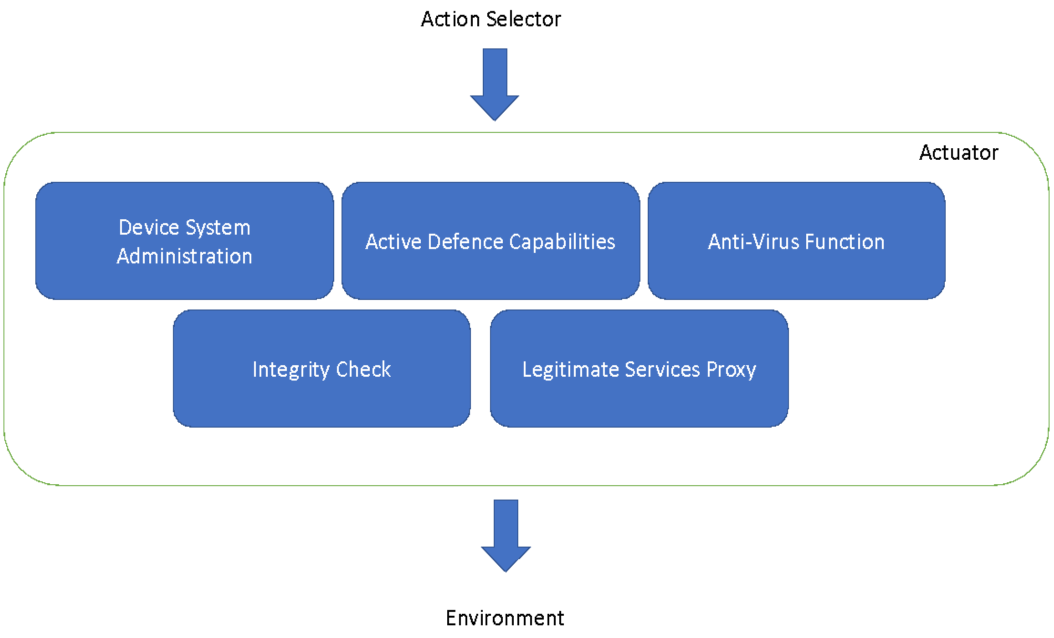
7.1 Overview

The overall purpose of the action execution component is to execute the action that the planning and action selector component has chosen according to the mission goals. The action execution component should be able to perform all the actions required to accomplish the typical tasks of a system administrator, including the security analysis of the system. This component has administrative privileges to execute the actions. To guarantee complete execution of the actions, the action execution component should run only atomic actions, either all the operations are completed or nothing occurs. In action execution, each action execution procedure takes as input the executable response plan sent from the action selector, and it outputs the response of the execution, which can be a message that confirms the successful execution or provides some details about the reason why the operation failed (e.g., if an action "Delete a file" fails, then the agent should provide to other agents some details like "The file cannot be deleted. The requested file does not exist." or "The file cannot be deleted. The requested file is protected.").

The action execution component is also connected with 2 other components: sensing and world state identification and learning. The sensing and world state identification component is connected with the action execution component because a sensor and an actuator, in practice, both represent an interface of the agent with the world. In this context, the only difference between them is that the actuator is able to change the world state, while the sensor is passive. In addition, the action execution component should be able to continuously update the internal rules and conditions with the feedback provided by the learning component.

7.2 Architecture

We design action execution as a container that is able to perform actions according to the situation. Action execution works as an actuator and is designed to perform a set of actions, as shown in Fig. 12.



Action Selector
Actuator
Device System Administration
Active Defence Capabilities
Anti-Virus Function
Integrity Check
Legitimate Services Proxy
Environment

Fig. 12 Overview of the action execution functionalities

Each category of actions has its own scope and conditions as explained in the following subsections.

7.2.1 Device System Administration

Device system administration deals with normal system operations, incident handling, and Root Cause Analysis. Overall, device system administration includes a set of actions that can be summarized as follows:

- install/remove software application
- software update
- registry modification
- user management
- log access
- baseline creation and periodic check sent to sensors

Based on the feedback that derives from the learning component, device system administration should be able to dynamically integrate new rules for each of the aforementioned set of actions.

7.2.2 Active Defense Capabilities

Active defense is a popular defense technique based on systems that hinder an attacker's progress by design, rather than reactively responding to an attack only after its detection. Since the goal of active defense system is to reduce the risk of a compromised system, in some cases, active defense can be used as a measure against lateral movements. Note that the purpose of active defense is not to defend or prevent the attacker from performing some actions. Instead, its goal is to slow the attacker down and allow optimal operation of the traditional defense systems.

The action execution component should be able to implement active defense capabilities to have the ability of perform annoyance, attribution, and, under some circumstances, even attack. The range of active defense actions can be described as follows:

- port remapping
- fake files
- fake services and network port
- fake web services
- fake supervisory control and data acquisition (SCADA) services
- attribution capabilities
- building a covert communication channel

The deployment of this set of actions enables a machine to act and react to the actions of an attacker, or an abnormal behaviour of a legitimate user, by slowing the adversary down with annoyance and attribution, and eventually, attack.

7.2.3 Antivirus Function

The action execution component should cover the antivirus function. This means that this component should be able to behave as an antivirus software, perform analysis, and not impact system functionality. For instance, this function should be able to execute the action "perform a full scan". The antivirus actions that the actuator should perform are the following:

- executable analysis
- complete device scan
- basic malware analysis heuristics.

The deployment of this set of actions in the actuator aims to enable the antivirus functionality of the machine and reduce the installation of antivirus software on the device itself. This means that when the device does not use an endpoint protection solution, the agent should be still able to guarantee the defense of the device. However, in the case when the device uses an endpoint protection solution, then the agent should be able to communicate and interact with this solution to take the necessary steps to quarantine, delete, or report the infected items. In this case, the antivirus function serves as a sensing function.

7.2.4 Integrity Check

Integrity check function evaluates the changes of the machine's state by periodically checking the stability and integrity of critical files that must not be changed without proper authorization. Depending on the configuration of the integrity check and the need of the action selector, the action execution component should check the integrity of the filesystem against both a whitelist and a blacklist. Since the integrity check is an action, it should be executed by an actuator, which then sends the data to the sensor to perform further analysis and updates the state of the world.

The sensor should consider the data that derive from the integrity check; therefore, the integrity check can serve as a starting alert to initiate the usage of the active defense capabilities. Also, the integrity check data can be used to build the system profile.

7.2.5 Legitimate Services Proxy

The legitimate services proxy should be implemented according to the active defense capabilities of the action execution and able to proxy any legitimate service of the host machine. The idea of legitimate services proxy is to use the actuator as a frontend interface toward the external environment, and then perform a security analysis of the incoming and outgoing traffic through the proxy. To support flexible active defense strategies, every legitimate port of the host machine should be bind to a port of the actuator, so that such port could be redirected to another port according to the active defense objectives. In other words, the proxy function should be able to expose a legitimate service to a nonstandard port without modifying the host machine.

7.3 Use Cases

7.3.1 Anomalous Behavior of a Military Vehicle

This attack scenario (Fig. 13) considers a compromised device that tries to probe the environment for information gathering. Some of the neighbor agents might recognize this anomaly, and they react by implementing an active defense capability. For example, the neighbor agents start to remap service ports or launch new fake services to profile the attacker. At the same time, the neighbor agents perform a scan on their local filesystem to check for suspicious executables and share the findings among themselves to build a real-time IoC. This operation involves a set of actions that should be executed on the infected device with administrator privileges and aims to restore the infected machine back to the normal state.

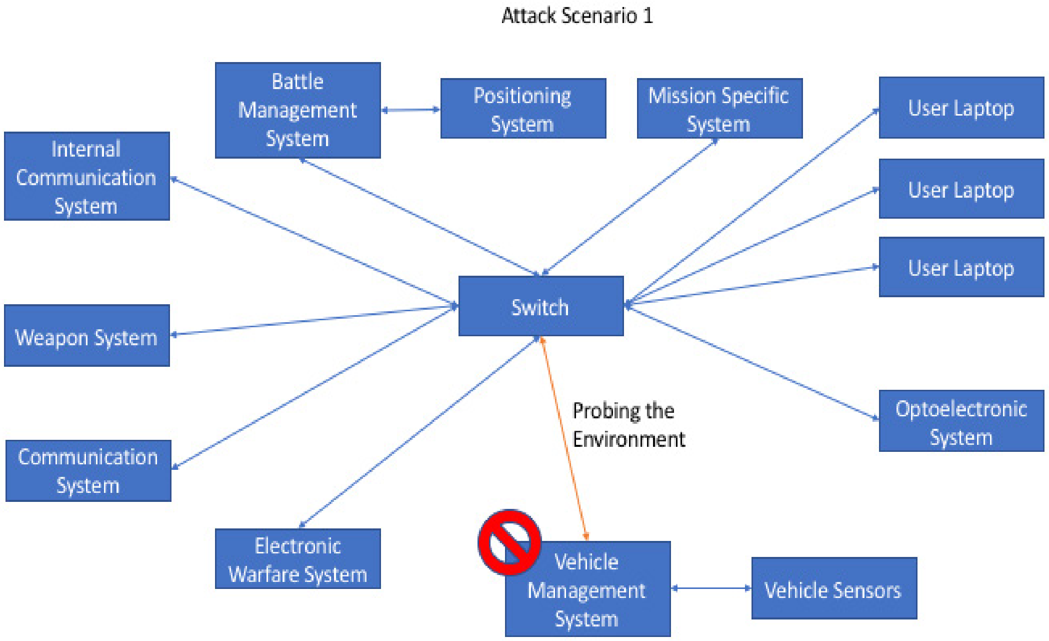


Fig. 13 Anomalous behavior of a military vehicle scenario

With active defense capabilities, agents will be able to perform an early detection based on abnormal behavior and reduce the risk of being persistently compromised by the attacker. The attacker is slowed down and the agent has more time to understand the attacker's goal.

7.3.2 Battle Management System, Vehicle Management System, and Communication System Compromised

In this scenario (Fig. 14), an agent that detects the compromise will create an uncompromised (covert) channel with other noncompromised agents to communicate the state of the situation and the possible compromised devices. The goal of this communication is to alert the other agents of the network not to trust the data that are coming from the compromised vehicle, avoid the transmission of sensitive data toward such vehicle, and also agree on a plan to recover the compromised devices in that vehicle.

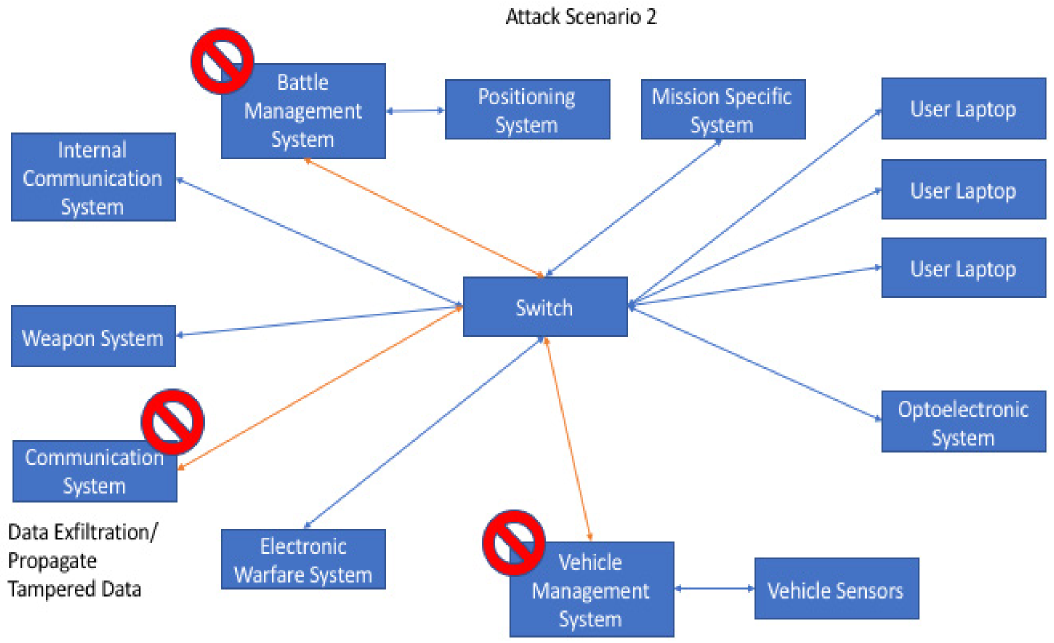


Fig. 14 BMS, VMS, and COMM compromised scenario

8. Collaboration and Negotiation

Authors: Edlira Dushku and Luigi Mancini

8.1 Overall Purpose

Battlefield operations are characterized by an unreliable communications infrastructure, limited network coverage, and also the presence of enemy forces that intend to compromise these operations. Considering these limitations, an intelligent agent that operates in a battlefield environment should be able to plan its own actions and possibly perform them in an autonomous way. However, under some conditions, a group of autonomous intelligent agents may need to collectively decide a joint plan of actions that solves a set of common goals. In this context, the collaborative agent model emerges as an effective approach that allows autonomous agents to collaborate and negotiate among themselves to accomplish their mission-critical goals and confront adversarial actions.

In the collaborative model of the AICARA, an agent can individually perform one or multiple tasks and also choose to cooperate with other agents to perform coordinated actions. Different from multiagent systems that aim to solve problems that are difficult or impossible for an individual agent to solve, in the collaborative agent system of AICARA, each individual agent should be able to solve the problem autonomously and only start to collaborate with other agents to improve the common plan of actions or extend the individual capacities of plan execution. In general, the interoperation between autonomous intelligent agents in AICARA intends to improve the active defense capabilities of the contested battlefields by enabling a collaborative decision-making process and improving the goal execution capabilities of individual agents.

Agent interoperability in AICARA is enabled by the collaboration and negotiation component, which coordinates the interactions agent–agent and agent–C2. Overall, the collaboration and negotiation component consists of 2 functions:

1) *Collaboration:* The collaboration function allows an individual Agent A to interact with other agents to make Agent A's plan of actions more effective or solve a task that is beyond Agent A's capabilities. Note that the information that Agent A perceives about the world state remains local and should not be shared with other agents. All the agents involved in a collaboration process should be able to exchange only the information related to their local plans.

2) *Negotiation:* The goal of the negotiation is to reach an agreement within a set of agents regarding a goal or a plan execution. During the negotiating process, agents agree in coordinating their plan of actions to reach a common goal.

8.2 Architecture of the Collaboration and Negotiation Component

The collaboration and negotiation function (Fig. 15) in the agent's structure should provide these fundamental services: 1) agent inquiry and discovery, 2) name discovery, 3) authentication, and 4) service and capacity discovery (SCD).

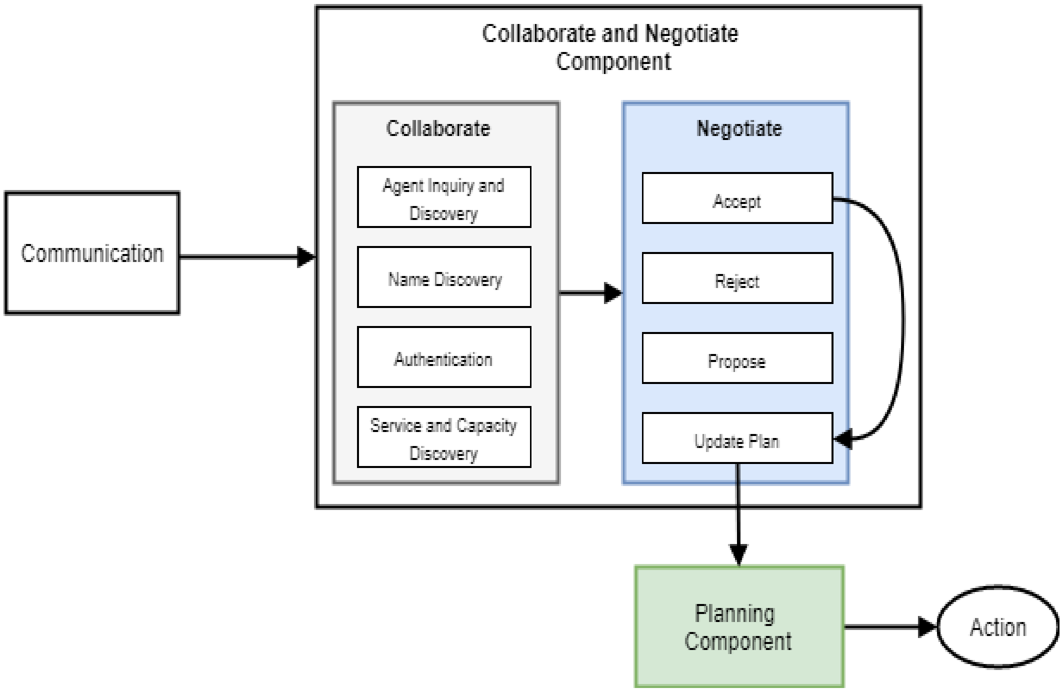


Fig. 15 Architecture overview of the collaboration and negotiation component

8.2.1 Agent Inquiry and Discovery

Agent inquiry and discovery is a procedure that allows an agent to be discovered by friendly forces. When a new agent joins the network, its presence can be detected by other agents and they can start collaborating. Under high-risk conditions, an agent can use this service to make a choice whether to be discoverable in the network or not. Likewise, when severe attacks are detected on the battlefield, the C2 unit can call this service to make agents undiscoverable from other agents.

8.2.2 Name Discovery

A procedure for retrieving the user-friendly name of a connectable agent. Friendly forces should share a common name taxonomy and should have some preshared cryptographic keys. The name of the agent should be connected to some configurations that agents know about each other. The agent process should be resilient to a Sybil attack. The agents that participate in the decision-making process should have the identity of the friendly forces. The enemy should not be able to influence the common goals.

8.2.3 Authentication

The collaboration function must enforce the authentication of the agents and provide data confidentiality and integrity of their communications. The authentication procedure describes how the required security is established when an agent initiates a collaboration request to a remote agent and when an agent receives a service collaboration from a remote agent.

8.2.4 Service and Capacity Discovery

SCD involves a set of procedures for querying and browsing the services offered by or through another agent. SCD does not define methods for accessing services; once services are discovered with SCD, they can be accessed in various ways, depending upon the service. After communication between 2 agents is established, they start exchanging information and computation. They also declare their capacities (memory, storage, CPU), which is very important in the later decision of allocating tasks to other agents.

The negotiation function consists of 4 services: 1) accept, 2) reject, 3) propose, and 4) update plan.

8.3 Collaborative Planning

Each agent has planning capabilities and can autonomously execute its local plan. An agent can extend its own planning capabilities by interacting with other agents to collaboratively construct a joint plan to accomplish their common mission goals.

When there is a new task/goal that the agent should achieve, the agent can choose to do the following:

1) Fulfill the task autonomously. In this case, the agent does not communicate with other agents and does not influence the goals of the other agents.

2) Distribute information about the task among the agents to reach a common plan of actions. This case requires several interactions among agents until they reach a common plan of actions. Since agents differ in capabilities and knowledge, they have different views regarding the task that should be fulfilled. During the interoperation, the data that an agent make accessible to other agents should present only the relevant information that is required for the collaborative planning and should not reveal sensitive information of the agent. This is important because an adversary, which could take control over an agent, should not be able to gain access to the sensitive information of other agents.

8.3.1 Communication Protocols

Agents can exchange information by using the following application protocols:

1) client-server:
   Simple Object Access Protocol (SOAP), Restful HTTP/Constrained Application Protocol (COAP)

2) publish-subscribe:
   Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Requested Power To Send (RPTS)

Agents should use protocols that guarantee the confidentiality and integrity of communications, for example, the basic security protocols such as Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS).

8.4 Decomposition Diagram

The collaboration and negotiation component allows an agent to discover other agents in the network and start to collaborate with them. Figure 16 captures the process flow of collaboration and negotiation activities.

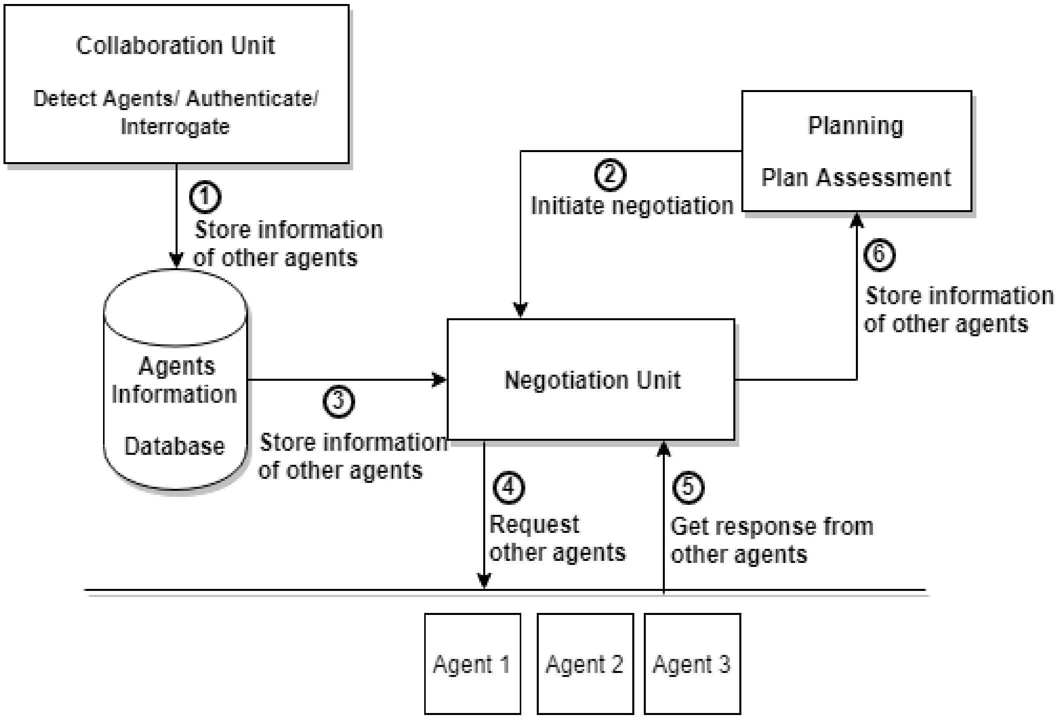

Collaboration Unit
Detect Agents/ Authenticate/ Interrogate
①
Store information of other agents
Agents Information Database
③
Store information of other agents
Planning
Plan Assessment
②
Initiate negotiation
Negotiation Unit
⑥
Store information of other agents
④
Request other agents
⑤
Get response from other agents
Agent 1
Agent 2
Agent 3

Fig. 16 Process flow in the collaboration and negotiation component

The process start with an agent that discovers other agents in the network. The collaboration between agents starts with a peer-to-peer authentication process.

After the authentication, neighboring agents interrogate among each other about the services and the capacities that they offer. Each agent saves in a storage all the information related to the other agents, as shown in Step 1 in Fig. 16.

The negotiation among agents is instantiated by the planning component. When the planned action is a complex task that requires more resource capacities than the autonomous agent can handle or the planned action affects the common plan of actions, then the planning unit decides to negotiate the plan of actions with other agents (Step 2). The negotiation unit retrieves all the agents' information from the database (Step 3) and then, based on the services and the resources that each agent offers, the negotiation unit requests the agent that satisfy the requirements of the planned action that should be executed (Step 4). Obviously, reasoning which agents can work on a planned action is a crucial factor for an effective collaboration among autonomous agents.

After sending the negotiation requests to some agents, the negotiation unit handles the responses that come from these agents (Step 5) and then forwards them to the planning unit (Step 6) for elaborating the next step of the plan execution.

8.5 Security Requirements

The autonomous agents deployed in battlefield should have a minimal Trusted Computing Base (TCB) to guarantee the trustworthy state of the agents. Also, the agents of friendly forces are expected to have some preshared cryptographic keys protected by hardware Root-of-Trust (RoT) on each agent. To guarantee a secure collaboration, all the services of the collaboration and negotiation component should use the security credentials embedded in the RoT.

The key management process should be based on a policy and should be performed by a specific security operation or an authority.

Since none of the distributed agents in the battlefield has a complete knowledge about the environment, it is important to construct the necessary security mechanism that could prevent a Sybil attack. The legitimate agents should be able to detect the agents with fake identity.

8.6 Use Cases

1) An Agent A coordinates with C2:

    a. Agent A detects a condition where C2's decision is needed (e.g., Application X1 in the sandbox behaves suspiciously).

b. Agent A sends a question to C2 (e.g., should I delete/kill application X1? Or else?).

c. C2 may reply or not.

    i. If C2 sends an authenticated reply to the agent, then Agent A performs the following steps:

        1. The agent receives the command sent from C2 (e.g., uninstall the application).

        2. The agent checks the feasibility of the execution (e.g., checks for permissions).

        3. The agent responds to C2 (e.g., "will do" or "cannot do, explain why").

        4. If "will do", Agent A generates plan and executes the plan (e.g., agent has to make a plan that checking all the dependencies, if there are some necessary services that are critical to be deleted, generate concealment plan).

        5. The agent sends the resulting state to C2 (e.g., respond "success" or "action failed, explain why").

    ii. If C2 does not give a response, then Agent A performs a local decision (e.g., agent decides what to do).

The flowchart of the interactions between Agent A and C2 is depicted in Fig. 17.

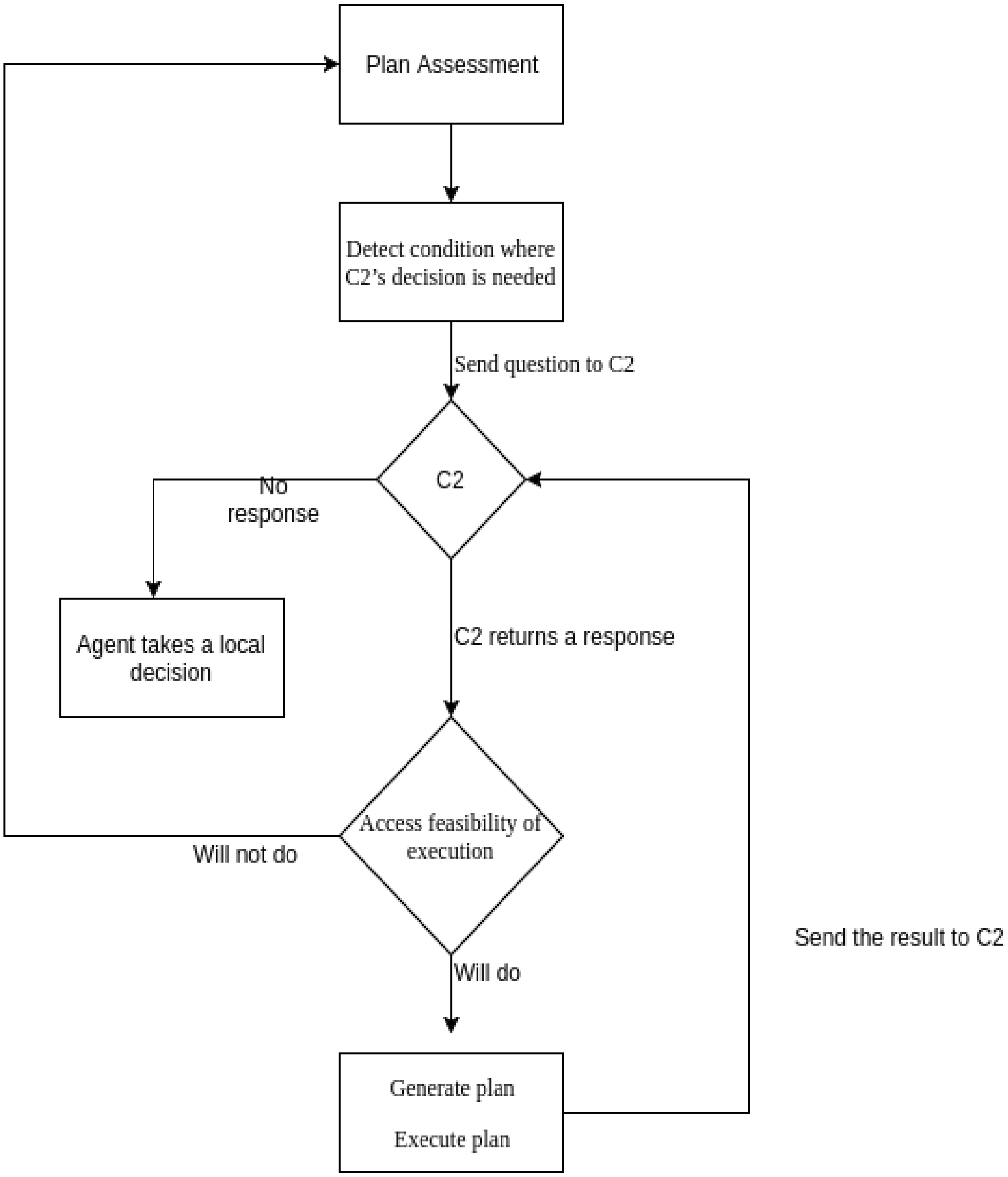


Fig. 17 Flowchart of interaction Agent–C2

2) Agent A collaborates with other agents:

   a. An agent communicates with other agents to improve the common plan of actions. For instance, if an Agent A identifies a malicious behaviour, Agent A notifies other agents and agree on changing their individual plans.

      i. Agent A identifies anomalous traffic caused from a malicious service S1.

      ii. Agent A detects the presence of Agent B nearby.

iii. Agents A and B establish communication and authentication, and declare services.

iv. Agent A notice that Agent B provides the same service S1.

v. Agent A notifies Agent B about the risk.

vi. Agent B gets the alert from Agent A.

vii. Agent B may perform one of the following actions:

1. Agrees to kill immediately service S1 that is running on the machine and evaluates again its local plan of actions considering the nonavailability of S1.
2. Completes the execution of the current plan and then kills S1.
3. Agrees to kill S1 if Agent A accepts to perform one of the action plan that Agent B must do.
4. Ignores the alert sent from A and continues its local plan.

b. In a similar way as the scenario explained previously, Agent A communicates with other agents to extend its local capacities in executing its individual plan of actions. For example, if a task needs to be executed and the resources are beyond the capacities of a single agent, then the task can be scaled to a group of agents:

i. Agent A realizes that the execution of an action X2 is taking a lot of time.

ii. Agent A detects the presence of Agent B nearby.

iii. Agents A and B establish communication and authentication, and declare services and capacities.

iv. Agent A notice that Agent B has the required capacities to perform the same action as A is running.

v. Agent A requests the Agent B to perform the action X2.

vi. Agent B gets the request from Agent A.

vii. Agent B may perform one of the following actions:

1. Agrees to immediately run X2.

2. Completes the execution of the current plan and then executes X2.
3. Rejects the request.

9. Learning

Lead Author: Alexander Kott

The environment of the agent can change rapidly, especially (but not exclusively) due to an enemy action. In addition, the enemy malware, its capabilities, and TTPs evolve rapidly. Therefore, the agent must be capable of autonomous learning. In this section, we offer examples of the vision of how the agent's learning could be implemented.

The reasoning capabilities (described in other sections of this report) of the agent rely on its knowledge bases (KBs). The purpose of the learning function(s) of the agent is to modify the KBs of the agent in a way that enhances the success of the agent's actions.

The agent learns from its experiences. Therefore, the most general cycle of the learning process is the following:

- The agent has a KB.
- The agent uses the KB to perform actions and also makes observations (receives percepts). These together constitute the agent's experience.
- The agent uses this experience to learn the desirable modifications to the KB.
- The agent modifies the KB.
- Repeat.

9.1 Representation of the Agent's Experience

At any time t, the agent performs action a, which could be a NULL action (i.e., there was no action); and perceives percept e, which also could be NULL. If the percept e, in conjunction with any prior information that the agent has, provides the agent with sufficient information to determine the state of the environment, and the closeness of that state to the goal state, then the agent may also be able to determine reward R. Otherwise, the reward is NULL.

Therefore, all the experiences of the agent can be represented with this sequence:

(t1, a1, e1, R1) (t2, a2, NULL, NULL) (t3, NULL, e3, R3) … (tn, an, en, Rn)

Here t1 is the time when the agent starts to record an experience and tn is the moment “now”.

To make the representation more compact and useful, we can divide it into shorter chunks; each chunk ends with a moment when the agent is able to determine the reward. We call such a chunk an episode. Episode Ej is a sequence of pairs {a1, ei}, and the resulting reward Rj:

Ej = ({ai,, ei},Rj)

The following is an example of a short episode:

- a1 checks filesystem integrity
- e1 finds unexpected file
- a2 deletes file
- e2 file gone
- a3 NULL
- e3 observes enemy C2 traffic
- Reward –0.09

Here is another example of an episode:

- a1 checks filesystem integrity
- e1 finds unexpected file
- a2 creates poisoned password file
- e2 NULL
- a3 NULL
- e3 receives alert from node 237
- Reward –0.57

9.2 What Can the Agent Learn?

What exactly is being learned? There are multiple options.

First, a fairly general option is that the learning module learns the world dynamics model. The world dynamics model is served by the data services module, but could be potentially generated or learned in other modules, particularly in the learning module. Generally, the world dynamics model is a function that takes as an input a state and an action applied to that state; its output is a new state that will result from application of that action or a distribution of states. This model could be predefined and then updated by the learning module. The world dynamics model is the same or very similar to function f_w, as discussed in the action selector and predictor module, which maps a world state and an action to a (set of) future world states (possibly with some information on the probability of individual states).

In addition, the learning module can optionally learn the f_a function discussed in the action selector and predictor module, which maps the current world state to a set of feasible actions

Furthermore, the learning module can optionally learn other elements:

- Reward for an action, possibly as a function of a sequence of prior actions and observations, or regardless of prior sequence. This is illustrated in Example 2 (Section 9.4) and Example 3 (Section 9.5).
- Reward for a sequence of actions (a plan), possibly as a function of a sequence of prior actions and observations, or regardless of prior sequence. This is illustrated in Example 1 (Section 9.3).
- Recommendation(s) of a suitable action(s) (i.e., a plan, possibly as a function of state, or of a sequence of prior actions and observations, or regardless of prior sequence). This is illustrated in Example 1 (Section 9.3).
- Classification of a sequence of observations as evidence of malicious activities of certain type.

9.3 Approach Example 1: Case-Based Reasoning

In this approach (Fig. 18), the learning is largely implicit. The agent collects its experiences in a collection of experiences, augments that collection by determining the state through which experiences passed, and determines rewards for those states, using a function called "state assessor". When the agent wants to determine a plan of actions, it looks at its most recent actions and matches them to the experiences. If a well-matching episode is found in its experiences and the resulting reward for that episode was sufficiently high, the agent use that episode as its plan for future actions.

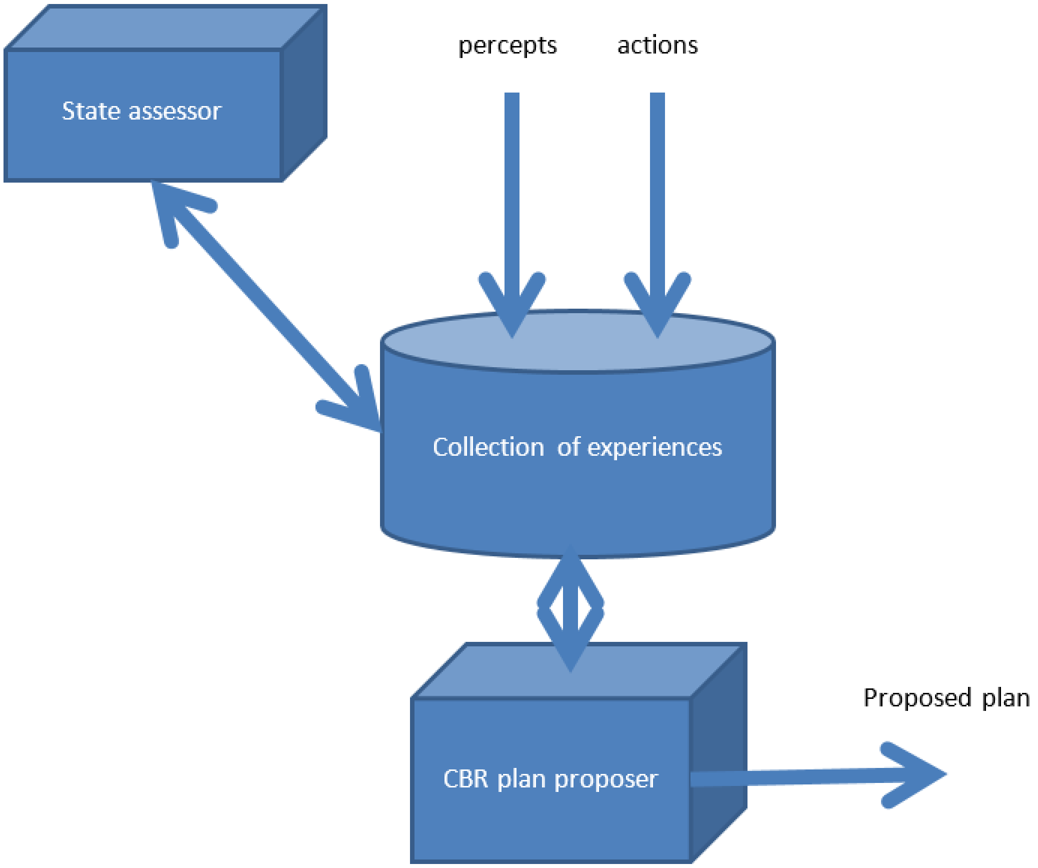


Fig. 18 Approach example 1: case-based reasoning

Consider the following, highly simplified illustration. Suppose the agent most recently took actions a13 and a76. The agent wants to formulate a plan of its next actions. The agent wants to make sure that the reward of the state that would result from its future actions should be at least 0.75. The agent accesses its collection of experiences and finds there the following episode: a13, a76, a06, a52, R = 0.83. The first 2 actions of that episode match the most recent actions taken by the agent. The reward is very good, higher than the 0.75. The agent, therefore, takes the remaining actions of that episode as its plan: it will proceed to execute actions a06 and a52.

Let us consider what, in this particular example, are inputs and outputs of the learning module.

Inputs include the following:

- Actions (each with a timestamp) that are provided most likely by the action execution module.

- Percepts (each with a timestamp), each of which is likely to be a change of state, arriving from the state model database.
- Goal state, as a predicate that defines whether a state is a goal state. This is provided by the data services.
- Distance function D(state, goal), which is a measure of how close the given state is to the goal. The lower is the distance, the closer is the state to the goal. When the distance is zero, then the state is a goal state. This function can also be interpreted as the reward function: the difference between D(state1, goal) and D(state2, goal) is the reward for moving from state1 to state2. This function is provided by the data services.

Outputs include the following:

- Updates to the collection of experiences, which serves as the primary KB. It can be made available to other modules, either directly or via the mediation of data services.
- Episode and the associated reward provided to the action selector and predictor modules.

Alternatively, if the agent has a separate planning function that generates plans, it can use its collection of experiences to predict the reward for executing that plan. For example, again suppose the agent most recently took actions a13 and a76. The planning function proposed a plan to execute actions a06 and a52.

The agent wants to know what the reward will be resulting from executing that plan. The agent accesses its collection of experiences and finds there the following episode: a13, a76, a06, a52, R = 0.83. The episode matches its past actions and the proposed future actions. Now the agent knows the reward is the proposed plan is executed: R = 0.83.

In this case, the inputs and outputs differ partially from the ones mentioned previously.

Inputs include the following:

- Actions (each with a timestamp) that are provided most likely by the action execution module.
- Percepts (each with a timestamp), each of which is likely to be a change of state, arriving from the state model database.
- Goal state, as a predicate that defines whether a state is a goal state. This is provided by the data services.

- Distance function D(state, goal), which is a measure of how close the given state is to the goal. The lower is the distance, the closer is the state to the goal. When the distance is zero, then the state is a goal state. This function can also be interpreted as the reward function: the difference between D(state1, goal) and D(state2, goal) is the reward for moving from state1 to state2. This function is provided by the data services.
- Plan provided by the action selector and predictor modules.

Outputs are the following:

- Updates to the collection of experiences, which serves as the primary KB. It can be made available to other modules, either directly or via the mediation of data services.
- Reward associated with the proposed plan, provided to the action selector and predictor modules.

Of course, this highly simplified illustration eschews many critical details: we did not mention anything about the percepts and states, and we did not discuss what to do when the match is not perfect. Nevertheless, the gist of the approach should be clear.

9.4 Approach Example 2: Deep Neural Network to Learn the Reward for the Next Action

This approach is inspired by the successes of deep mind (Mnih et al. 2013). Similarly to deep mind, here our agent uses the collection of experiences to train a neural network. The inputs are the actions and percepts for a number of previous time points. The outputs are, for each possible action of the agent, the reward associated with taking that action as the next action. Once the neural network is trained, it is used at each time point to determine the next action—the one with highest reward.

To explain how the neural network might look like, consider a highly simplified example. Suppose, at any given time, the agent can take one of only 3 actions: a1, a2, and a3. (In a practical implementation, there could be thousands of possible actions.) At any given time, it can receive one of only 4 percepts: e1, e2, e3, and e4. (In practical implementations, there could be thousands of possible percepts.) In our neural network, we consider only 2 time points: the most recent time an action was taken and the previous time point. (In a practical implementation, multiple time points could be considered.) Figure 19 depicts the neural network after it has been trained. At the most recent time, the agent has performed

Action a2 and received Percept e3. Right before that, it performed a3 and perceived e1. These are the data that go into the input layer. The neural network uses these inputs to produce the outputs: if the next action taken by the agent is a1, the reward will 0.07, if the next action is a2, the reward will be 0.023, and if the next action is a3, the reward will be 0.79. Naturally, the agent will select a3, the one with the highest reward.

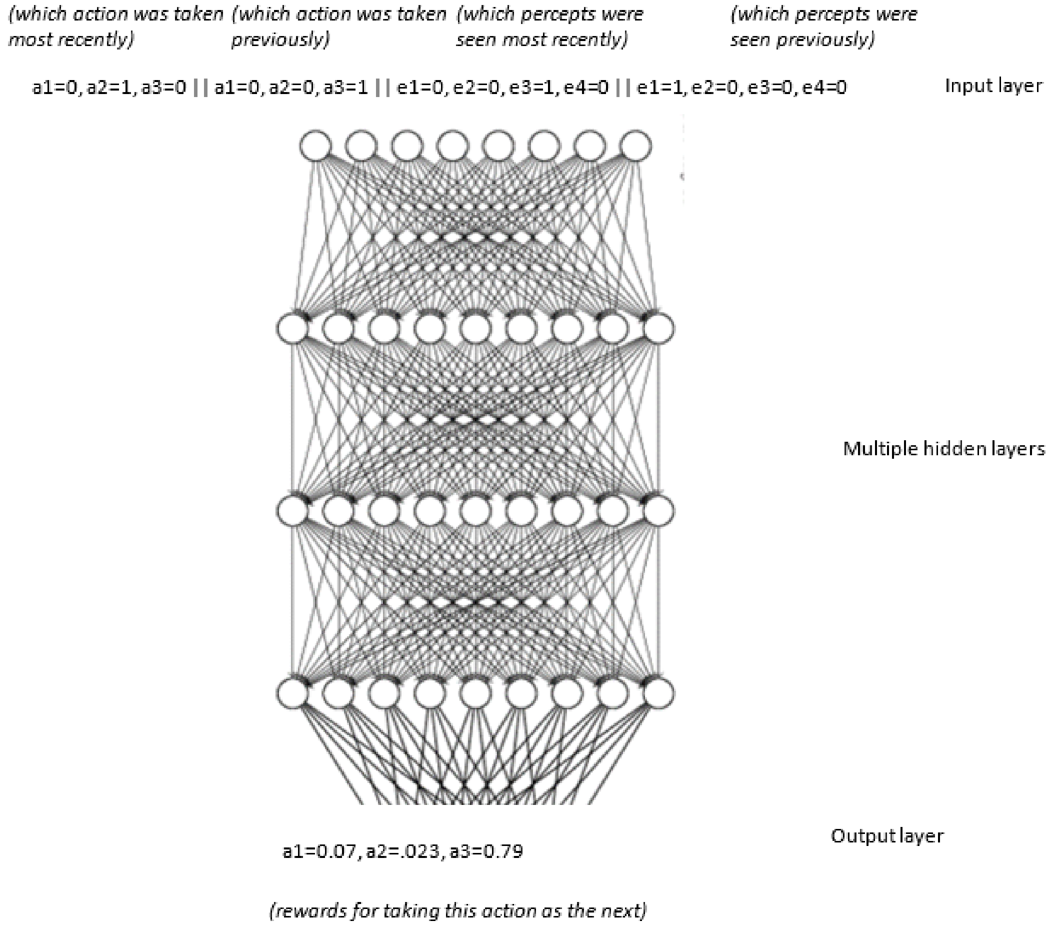

(which action was taken most recently)
(which action was taken previously)
(which percepts were seen most recently)
(which percepts were seen previously)
a1=0, a2=1, a3=0 || a1=0, a2=0, a3=1 || e1=0, e2=0, e3=1, e4=0 || e1=1, e2=0, e3=0, e4=0
Input layer
Multiple hidden layers
Output layer
a1=0.07, a2=.023, a3=0.79
(rewards for taking this action as the next)

Fig. 19 Neural network after it has been trained

The architecture of this approach is illustrated in Fig. 20.

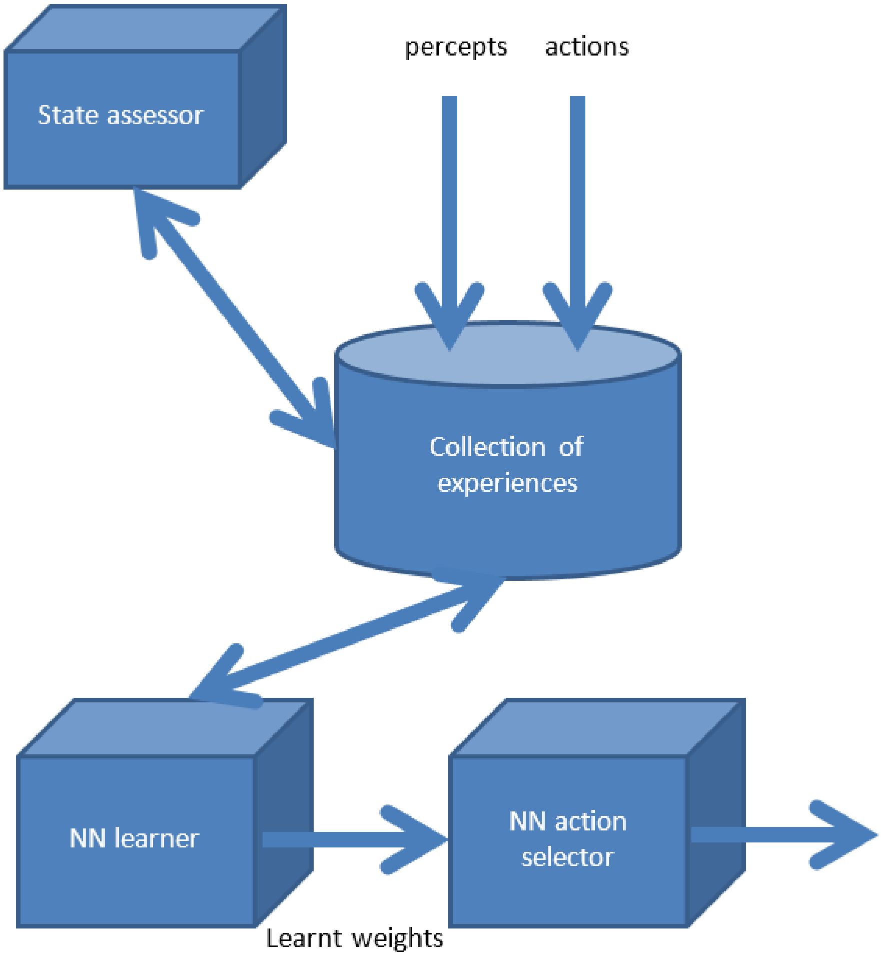


Fig. 20 Approach example 2: deep neural network to learn the reward for the next action

Let us consider what, in this particular example, are the inputs and outputs of the learning module.

Inputs include the following:

- Actions (each with a timestamp) that are provided most likely by the action execution module
- Percepts (each with a timestamp), each of which is likely to be a change of state, arriving from the state model database
- Goal state, as a predicate that defines whether a state is a goal state. This is provided by the data services.
- Distance function D(state, goal), which is a measure of how close the given state is to the goal. The lower is the distance, the closer is the state to the goal. When the distance is zero, then the state is a goal state. This function can also be interpreted as the reward function: the difference between

D(state1, goal) and D(state2, goal) is the reward for moving from state1 to state2. This function is provided by the data services.

Outputs include the following:

- Updates to the weights of the neural net, which serve as the primary KB
- The best next action and the associated reward provided to the action selector and predictor modules

Note that Mnih et al. (2013) used a deep neural network as a component of a Q-learning approach (Watkins et al. 1992). Indeed, Q-learning is very appropriate in such problems as ours. It is extremely unlikely that a sufficiently complete model (i.e., probabilities of state transitions given an action) can be constructed for operations of a computer or a network of computers. Therefore, the only option is to pursue some form of model-free reinforcement learning; this means Q-learning, that is, action-value learning.

9.5 Approach Example 3: Simplified Statistical Learning of the Action Reward

It may be worthwhile to explore a very simple, low-cost approach for learning the reward for taking an action. Suppose, for each action ai, the learning algorithm collects statistics (from the collection of episodes) on how often the action ai is a part of an episode that leads to a relatively high reward value and how often it leads to a low reward. The algorithm uses a heuristic to allocate the reward to the actions participating in the episode (e.g., allocate 50% of the reward to the most recent action, and distribute the rest of the reward equally among the preceding 5 actions). As experience accumulates, each action's reward is updated. Whether such a simplistic learning algorithm can be useful is a matter of empirical research. Even if not particularly efficacious, it might be suitable for very lightweight agents.

A similar approach could be applied to n-gram actions, instead of a single action.

10. Conclusion

Intelligent, partly autonomous agents are likely to become primary cyber fighters on the future battlefield. Our initial exploration identified the key functions, components and their interactions for a potential reference architecture of such an agent.

With respect to further related efforts, the current priorities are the following:

- To study use cases as a reference for the research, as this will lead to clarifying the scope, concepts, functionality, and functions' inputs and outputs of AICA-like systems.
- To refine the initially assumed architecture by drawing further lessons from the case studies.
- To determine the set of technologies that the AICAs should embark on and that need to be tested during the prototyping phase.
- To define the methodology of the tests.

The sum of challenges presented by the AICA concept appears, today, very substantial, although our initial analysis suggests that the required technical approaches do not seem to be entirely beyond the current state of the research. An empirical research program and collaboration of multiple teams should be able to produce significant results and solutions for a robust, effective intelligent agent. This might happen within a timespan that could currently be assumed on the order of 10 years.

11. References

De Gaspari F, Jajodia S, Mancini LV, Panico A. AHEAD: a new architecture for active defense. SafeConfig'16; 2016 Oct 24; Vienna, Austria.

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. CoRR. 2013:1312.5602. arXiv preprint arXiv:1312.5602.

Muttik I. Good viruses. Evaluating the risks. DEF CON 16; 2008 Aug 8-10; Las Vegas, NV. https://www.defcon.org/images/defcon-16/dc16-presentations/defcon-16-muttik.pdf.

Russell S, Norvig P. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River (NJ): Prentice Hall; 2009.

Watkins CJCH, Dayan P. Q-learning. Mach Learn. 1992;8(3-4):279–292.

Bibliography

Boddy MS, Gohde J, Haigh T, Harp SA. Course of action generation for cyber security using classical planning. ICAPS. 2005 June:12–21.

Korzhyk D, Yin Z, Kiekintveld C, Conitzer V, Tambe M. Stackelberg vs. Nash in security games: an extended investigation of interchangeability, equivalence, and uniqueness. J Art Int Res. 2011 May;41(2):297–327.

Kott A, Alberts DS, Wang C. Will cybersecurity dictate the outcome of future wars? Computer. 2015;48(12):98–101.

Kott A, Singh R, McEneaney WM, Milks W. Hypothesis-driven information fusion in adversarial, deceptive environments. Inform Fusion. 2011;12(2):131–144.

Kott A. Towards fundamental science of cyber security. In: Pino RE, editor. Network science and cybersecurity. New York (NY): Springer; 2014.; p. 1–13.

Rasch R, Kott A, Forbus KD. AI on the battlefield: an experimental exploration. Proceedings of the 14th Innovative Applications of Artificial Intelligence Conference; Edmonton, AB; AAAI/IAAI; c2002.

Rasch R, Kott A, Forbus KD. Incorporating AI into military decision making: an experiment. IEEE Int Sys. 2003;18.4:18–26.

Sarraute C, Buffet O, Hoffmann J. POMDPs make better hackers: accounting for uncertainty in penetration testing. 26th AAAI Conference on Artificial Intelligence; 2012.

Sarraute C, Richarte G, Obes JL. An algorithm to find optimal attack paths in nondeterministic scenarios. Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence. ACM; 2011.

Stytz, Martin R., Dale E. Lichtblau, and Sheila B. Banks. Toward using intelligent agents to detect, assess, and counter cyberattacks in a network-centric environment. Alexandria (VA): Institute for Defense Analyses; 2005.

Van Dijk M, Juels A, Oprea A, Rivest RL. FlipIt: the game of “stealthy takeover”. J Crypto. 2013;26(4):655–713.

List of Symbols, Abbreviations, and Acronyms

AICA	Autonomous Intelligent Cyber Defense Agent
AICARA	Autonomous Intelligent Cyber Defense Agent Reference Architecture
AMQP	Advanced Message Queuing Protocol
APT	advanced persistent threat
ARL	US Army Research Laboratory
BMS	battle management system
C2	command and control
C4ISR	command, control, communications, computers, intelligence, surveillance, and reconnaissance
CAPEC	Common Attack Pattern Enumeration and Classification
COAP	Constrained Application Protocol
COMM	communication system
CPU	central processing unit
CVE	Common Vulnerabilities and Exposure
DTLS	Datagram Transport Layer Security
EW	electronic warfare
f_a	set of possible plans of actions
f_w	set of feasible actions
HTTP	Hypertext Transfer Protocol
HQ	headquarters
InterCOM	internal communication system
IoC	indicators of compromise
IT	information technology
KB	knowledge base
MISP	Malware Information Sharing Platform

MQTT	Message Queuing Telemetry Transport
MS	mission-specific system
NATO	North Atlantic Treaty Organization
OPT	optoelectronic system
OS	operating system
PBS	packet-based switching
RoT	Root-of-Trust
RPTS	requested power to send
RTG	Research Task Group
S	sensors
SCADA	supervisory control and data acquisition
SCD	service and capacity discovery
SHA-1	Secure Hash Algorithm **1**
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SW	switch
TCB	Trusted Computing Base
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTPs	tactics, techniques, and procedures
VMS	vehicle management system
WS	weapon system
WSI	world state identifier

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HRA RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	ARL RDRL D A KOTT

INTENTIONALLY LEFT BLANK.